General Information

The following sections describe general information about bidirectional language support.

About this Book

The ***Bidirectional Language Programming Guide*** provides information about the Presentation Manager (PM) application programming interfaces (API) for the Arabic and Hebrew languages that let programmers create Arabic or Hebrew applications for this environment.

The Bidirectional support in PM is provided in the Arabic and Hebrew language versions of OS/2. The support is active/configured when the COUNTRY selection during OS/2 installation is Arabic or Hebrew in these versions.

Introduction

The purpose of this guide is to provide the necessary information to allow the creation of applications that work in Arabic or Hebrew. Reading and writing of Arabic or Hebrew text is done from Right-To-Left. This means that the "start" point of a Hebrew/Arabic text is on the right side of the page/screen while the natural end of the text is on the left side.

In many cases, Arabic and Hebrew scripts contain embedded pieces of text which are written and read from Left-To-Right. In fact, having mixed Right-To-Left (RtL) and Left-To-Right (LtR) "segments" of text is very common in these languages. Examples of such cases is the use of numbers (which are usually LtR, as in English) and the use of embedded Latin text within Arabic/Hebrew text, which is also very common.

Having such text, with embedded Right-To-Left and Left-To-Right pieces (segments), is the reason for referring to the Arabic and Hebrew languages as "Bidirectional Languages" (or bidi languages, for short).

In this document, we will use the term 'National Language' or 'Bidi Language' to refer to 'Arabic or Hebrew'. (to distinguish it from English). Also, 'Arabic/Hebrew' is used to signify 'either Arabic or Hebrew'.

Bidirectional Applications

Bidirectional applications are applications that support the Arabic and Hebrew languages. By nature, bidirectional applications have to be sensitive, to some extent, to the fact that their text data is bidirectional (contains embedded LtR and RtL segments).

In this context there are two types of applications:

- Bidi-Aware Applications

  These applications are written specifically to be used by Arabic and Hebrew-speaking users. These applications are aware of the fact that the text being processed is bidirectional, and take steps to ensure the correct processing of this text. Bidi-aware applications make active use of the PM bidirectional support APIs.

- Bidi-Unaware Applications

  Bidi-Unaware applications do not have any knowledge about the fact that the text they handle/process is bidirectional. Usually, these applications are developed for users of a Left-To-Right language (such as English or French), and thus they do not perform any special handling of bidirectional text. Such applications do not make explicit calls to the PM APIs for bidirectional support.

Scope of Bidirectional Support

The PM bidirectional support has been designed with the following main design objectives:

- Allow creation of bidi-aware applications.
- Allow as easy as possible conversion of bidi-unaware applications to being bidi-aware.
- Allow usage of bidi-unaware applications in the National Language. The extent to which this is possible is largely dependent on the application itself.

Thus, the full bidirectional support is provided to applications which make active use of the PM Bidirectional services (APIs, PM messages etc). In addition, a certain degree of support is also provided for bidi-unaware applications. Depending on the application, this support (for bidi-unaware) may provide various degrees of usability for the application when working with Arabic or Hebrew.

The Bidirectional support in PM is provided in the Arabic and Hebrew language versions of OS/2. The support is active/configured when the COUNTRY selection during OS/2 installation is Arabic or Hebrew in these versions.

Making an Application Bidi-Aware

Most of the applications that are developed today throughout the world are created as Bidi-unaware applications (with the exception of applications developed in the countries where the native language is Arabic or Hebrew). Such applications can be made bidi-aware by adding calls to the PM bidi support APIs and by adding code that is aware of bidirectional text and knows how to manage it.

The PM bidirectional support provides the following types of services/mechanisms to assist the conversion process of an application to be bidi-aware.

- Bidirectional text presentation

  Support for bidirectional text presentation is available. This support is sensitive to the contents of the BIDIATTR environment variable. When this variable is set, the text output functions do some extra processing to the text being output. This support provides very basic bidirectional support for the benefit of bidi-unaware applications. For applications that use text in a very simple way (i.e they do not have any sophisticated text handling features), enabling this support (by initializing the BIDIATTR environment variable) may be enough. For this basic support, the application code does not need be changed at all. The fact that the unaware application runs in a bidi-supporting environment (denoted by the BIDIATTR variable) is enough to trigger the system-provided bidirectional text handling and thus gain some level of automatic support for such applications.

  In most cases, however, some additional support will be needed. In these cases, some changes must be made to the application in order to allow it to support bidi text.

- Bidirectional Input support

  This support allows typing of characters in both English and Hebrew/Arabic, as well as support for language-specific hotkeys. This is the basic means that allows the user to input national language text from the keyboard.

- Bidirectional PM Controls.

  PM applications make extensive use of system services called PM Controls. These services provide support for many routine tasks such as text entry, menu management, selections from a list of options and scrolling.

  With the PM Bidirectional support, all PM controls are bidi-aware services. Applications can make use of these bidi-aware services by marking them as bidi-aware and specifying their bidirectional attributes. This way, an application can be made aware (up to a certain extent) just by marking its PM resources as Bidi resources. In many cases, this process involves only changes in resource files (.RC) rather than changing source (and executional) code.

- Bidirectional support APIs.

  The third level of support is available only for applications that incorporate in their code explicit calls to the various PM Bidirectional APIs, such as calls for text conversions, keyboard input support, justification etc.

Supported Code Pages

The PM bidirectional support is dependent on the active codepage being used by the application. In most cases, the system determines what actions are taken and what type of processing is done based on the active codepage.

In this version of PM, the codepages that are supported for bidirectional processing are:

- Codepage 862 - Hebrew PC.
- Codepage 864 - Arabic PC.

Arabic and Hebrew Fonts

The Arabic and the Hebrew codepages are bilingual (i.e. they contain English and Arabic or English and Hebrew characters). When either of these codepages is active, the system fonts support all the characters in the relevant codepage. This means that the system fonts are bilingual, for example "Courier" would contain both English and Arabic/Hebrew. shapes.

This means that applications do not need to change fonts when switching between Arabic/Hebrew and English characters.

Note: Arabic and Hebrew postscript (Adobe Type 1) fonts are created using font specific encoding. This means that the value used for the codepage of these fonts is "65400" rather than the national codepage. Applications that specify the usCodepage field when creating a logical font should use this value in order to get a match for the Bidi postscript fonts.

Bidirectional Text and Attributes

The following sections describe bidirectional text and attributes.

About Bidirectional Text

To correctly process bidirectional text, applications must assume that text is stored in computer memory in a "different way" from when presented on an output device. In other words, a text-conversion process of some sort takes place when bidirectional text is retrieved from storage (files, memory) to be displayed on the screen (or printer), or when it is read from the keyboard and stored.

There are many ways to represent (store in memory, or files) bidirectional text that contains Arabic or Hebrew characters, English characters and numbers (digits). Different applications use different methods to store their text data and use different conversions before displaying their text.

Applications may choose to use different formats to represent bidirectional text, depending on the nature of the processing that they perform on this text.

In order to support bidirectional text processing, the PM support for Hebrew and Arabic defines a set of parameters that describe the different "features" that any Arabic or Hebrew bidirectional text may have. These parameters are called "Bidirectional Attributes".

A bidirectional application can describe the nature of the bidirectional text data it processes, by identifying a set of bidirectional attributes. These bidirectional attributes, which are implemented in PM in a hierarchical model, uniquely define the bidirectional text and the conversions it must go through, to the PM system and to the PM bidirectional services.

Below is the list of the bidirectional attributes as well as a short explanation of each one of them.

Bidirectional Attributes

The following sections describe the bidirectional attributes.

Text Type

The 'Text Type' bidi attribute defines the method by which text segments (that have a certain direction) are detected and processed. The text type attribute can have the following values:

- Visual text

  The 'Visual' data type means, that for a certain piece of text there are no sub-segments. In other words, the whole text has the same direction (whether it is LtR or RtL is determined by the Text Orientation attribute, see below). Whether the text contains digits, English, or Arabic/Hebrew characters does not change the sequence of the characters within this given piece of text.

  For example (LTR orientation; upper case letters represent English letters, while lowercase letters represent Arabic or Hebrew letters):

```
  Index of character: 012345678901
  Storage buffer    : LATIN cibara
  Displayed String  : LATIN cibara

or, in the case of RtL orientation:

  Index of character: 012345
  Storage buffer    : arabic
  Displayed String  : cibara
```

- Logical text with implicit detection and processing.

  This value is sometimes called "implicit" (for short) or "typing order" as it usually represents the order of characters as they are typed on the keyboard.

  Here, the determination of the sub-segments is largely based on the actual characters in the text. When there is a doubt (as in the case of characters which do not have a "strong" language meaning associated with them, such as SPACE, QUESTION MARK and digits), the determination of the sub-segments and the conversion required is done by a "smart" algorithm.

  Example (LTR orientation; upper case letters represent Latin letters, while lowercase letters represents Arabic or Hebrew letters).

```
  Index of character: 012345678901
  Storage buffer    : LATIN hebrew
  Displayed String  : LATIN werbeh
```

Text Orientation

This attribute defines the orientation of a text string (object). It may have the following values:

- Left-To-Right
- Right-To-Left
- Contextual

For text that has the Visual data type, the text orientation determines the order of the characters. For example, when visual text has Left-To-Right orientation, the order of the characters in the memory buffer and the presentation order (from left-to-right) is the same.

For text that has the Logical/Implicit data type, the text orientation of a certain piece of text defines which type of characters are considered the "base" text. The characters which belong to the "base" text retain their sequence when converting from storage to presentation or vice versa, while the characters which belong to embedded segments ("non base" text) are reversed.

For LtR text, Latin text retains its order while Arabic/Hebrew text is reordered (reversed).

For RtL text, Arabic/Hebrew text retains its order while Latin text is reordered (reversed).

Contextual text orientation is not exactly an orientation value (obviously, orientation can be only LtR or RtL). The contextual text orientation value tells the system that the actual orientation of the given text should be determined based on the first "strong" character in the text (in storage buffer indexes). I.e, if the first character in the buffer is an English character, the orientation of the text will be Left-To-Right. If the first character in the buffer is Arabic/Hebrew
- the text orientation is Right-To-Left.
(If the first character in the buffer is a "neutral" character, such as a SPACE, the orientation is determined by the next character in buffer, and so on).

Window Orientation

The "Window Orientation" bidi attribute is unique among all other bidi attributes, as it defines a property of a WINDOW (rather than a property of the text being processed). The window orientation attribute is "packaged" in the bidi attribute word for convenience, since it is a very handy placeholder for bidi-oriented information.

The Window Orientation attribute covers several visual aspects of a window. These include:

- ***Layout***

  In the Window Orientation context, layout refers to the positioning of objects within a window or dialog.

  A window with a LtR window orientation has its origin at the bottom left.

  A window with a RtL window orientation has its origin at the bottom right of the window instead of the bottom left, thus placement of objects (e.g. child windows, controls, vertical scrollbar, etc.) within the window will be in the symmetrically opposite position to those in a LtR window.

  Some examples of how this works:

  - RtL Buttons have their text positioned to the left of the 'button bitmap'.
  - RtL Horizontal scrollbars have their origin on the right.
  - Vertical scrollbars in listboxes, MLEs, etc. are placed on the left side of the window.
  - This concept is applied in the same way to other objects in the system.

- ***Justification***

  In many cases the default justification of objects within a Right-To-Left object is on the right.

In general, a window with RtL window orientation has Right-To-Left layout and its objects are right-justified.

Symmetric Swapping

Directional characters are characters such as parentheses, brackets, braces, etc. In LtR text, an "open" bracket is a left bracket, while for RtL text an "open" bracket is a right bracket.

Sometimes (especially in the process of transforming a bidi-unaware application to be bidi-aware), it is useful to have the system automatically "swap" between the symmetric characters. This way, the processing part of an application may be left unchanged, while the presentation part is taken care of by this automatic swap. Note that this automatic swap is relevant only for RtL text (or RtL text segments within LtR text).

For Example (Visual, RtL text).

```
Storage  buffer  : (1) hebrew
Display          : werbeh )1(       without symmetric swapping
Display          : werbeh (1)       with symmetric swapping
```

Character Shape Determination

Arabic is a cursive script. This means that the presentation of each letter depends on its location in the word (initial, middle, final or isolated).

Data may be stored "shaped", i.e. each letter of the text is stored in the specific codepoint (out of the different possible codepoints for that letter) that suits its position in the word and the connecting capabilities of its adjacent letters. Such data can be displayed directly.

Data may also be stored in nominal or base shapes, i.e. one codepoint per letter, regardless of its position. Such data must be "shaped" before it is displayed.

The text shape attribute defines how the Arabic text is stored and also what type of conversion should be performed on it in order to display it correctly. It can take the following values:

- TEXT_DISPLAY_SHAPED

  This implies that the text is stored in nominal/base shapes and should be automatically shaped upon display.

  PM entryfields that have this attribute automatically shape new text as it is being entered by the user and will submit the text in nominal shapes upon query.

- TEXT_SHAPE_NOMINAL

  This implies that the text is displayed and stored in its nominal (base) shapes.

  This text shape is not commonly used.

  The following shaping modes are valid only when the type of text is visual:

  - TEXT_SAVE_SHAPED

    This implies that text is already shaped and ready for display.

    PM entryfields that have this attribute automatically shape new text as it is being entered by the user and will submit the text as it appears on the screen upon query.

  - TEXT_SHAPE_INITIAL

    This implies that text is already shaped and ready for display.

    PM entryfields that are set to this mode shape new characters typed by the user in initial shapes and will submit the text as it appears on the screen upon query.

    This mode is usually initiated with a hotkey by users who need to type some characters specifically in their initial shapes.

  - TEXT_SHAPE_MIDDLE

    This implies that text is already shaped and ready for display.

    PM entryfields that are set to this mode shape new characters typed by the user in middle shapes and will submit the text as it appears on the screen upon query.

    This mode is usually initiated with a hotkey by users who need to type some characters specifically in their middle shapes.

  - TEXT_SHAPE_FINAL

    This implies that text is already shaped and ready for display.

    PM entryfields that are set to this mode shape new characters typed by the user in final shapes and will submit the text as it appears on the screen upon query.

    This mode is usually initiated with a hotkey by users who need to type some characters specifically in their final shapes.

- TEXT_SHAPE_ISOLATED

  This implies that text is already shaped and ready for display.

  PM entryfields that are set to this mode shape new characters typed by the user in isolated shapes and will submit the text as it appears on the screen upon query.

  This mode is usually initiated with a hotkey by users who need to type some characters specifically in their isolated shapes.

Numerals

The numerals attribute detemines how numerals are processed upon display as follows:

- NUMERALS_NOMINAL - Display only nominal (i.e. Arabic) numerals.
- NUMERALS_NATIONAL - Display only national (i.e. Hindi) numerals.
- NUMERALS_CONTEXT - Display numerals according to surrounding text.
- NUMERALS_PASSTHRU - Display numerals as stored.

Conversions of Bidirectional Text Data

Bidirectional applications use bidirectional text. The bidirectional text is described by bidirectional attributes. Since applications use different sets of bidirectional attributes (and thus, different representations of bidirectional text), it is necessary to convert this text among representation forms. The most common case is when an application converts the text from the internal (storage) representation to the external (presentation) form.

OS/2 provides a set of APIs to manage the conversion of bidirectional text between different formats. The text is assumed to be in a *form/layout* that is described by a set of input values and is converted to another form described by a set of output values.

In order to perform a text conversion, the caller must first call LayoutCreateObject to create a layout object. The layout object is automatically associated with a set of default values, known as layout values, that depend on the locale used to create it.

Currently, the following locales are supported:

- Locale_Arabic
- Locale_Hebrew

Using the handle to the layout object created, the caller can query the current values using LayoutQueryValues and set them, using LayoutSetValues, to the values required for the text transformation.

The actual text transformation is done by the LayoutTransformText API, which converts its input buffer according to the values associated with the layout object. LayoutTransformText can also optionally provide some additional information about the transformation performed, such as mappings of the input buffer to the output buffer.

LayoutEditShape is a more specialized text conversion API that allows an application to do Arabic character shape editing in and around a specific location in a buffer.

When an application has finished its text conversions, it should release the resources used by the layout object using LayoutDestroyObject.

See more on layout values and programming in the reference section.

PM Support for Bidirectional Attributes

Presentation Manager provides support for bidirectional text by providing applications with APIs to enable them to manage bidirectional attributes at several levels.

- Process Bidirectional Attributes
- Window Bidirectional Attributes
- Presentation Bidirectional Attributes
- Clipboard Bidirectional Attributes

Process Bidirectional Attributes

Process bidirectional attributes are used to assign applications a default set of attributes. These attributes are used by various PM components to determine the processing that must take place.

In this version, the Process bidirectional attributes are determined based on value of the "BIDIATTR" OS/2 environment variable which is in effect at the time the program is loaded into memory. The BIDIATTR environment variable can have multiple values that represent bidirectional attributes. For example:

```
SET BIDIATTR=TEXT_ORIENT_RTL,TEXTTYPE_VISUAL,TEXT_SAVE_SHAPED
```

The names of the various bidirectional attributes are documented in Appendix D.

Applications can set the process bidirectional attributes by using the WinSetLangInfo API, with the parameter LI_BD_PROCESS_ATTR.

Window Bidirectional Attributes

Window Bidirectional Attributes are assigned to each window in the Presentation Manager system. Window bidirectional attributes are usually inherited by a window, at creation time, from its parent window. This is true for all windows, except for windows that are children of the desktop window. These windows inherit their window bidirectional attributes from the process bidi attributes.

Applications use the WinSetLangInfo API with the LI_BD_WND_ATTR parameter to set specific window bidi attributes. Note that with the LI_BD_WND_ATTR applications have control on the way bidirectional attributes are inherited within the parent-child chain.

PM controls (which are part of the PM system) are sensitive to their bidirectional attributes. They determine their window layout based on the 'window orientation' attribute and perform text conversions based on other bidirectional attributes.

Presentation Bidirectional Attributes

These bidirectional attributes are associated with text output operations. The text output APIs in PM (WinDrawText and the various GpiCharString APIs) are sensitive to the presentation bidirectional attributes. These attributes are inherited from the process bidirectional attributes at creation time of each GPI presentation space.

The text-output APIs use the presentation bidi attributes to determine the conversions that must be applied to the text as part of the processing of the text-output APIs. For example, GpiCharString will convert bidirectional text, by calling the LayoutTransformText API on behalf of the application program. The bidi-converted string will then be rendered on the the output device (rather than the original string that has been provided in GpiCharString API call).

Applications can optionally modify the presentation bidi attributes dynamically by using the GpiSetBidiAttr API. Thus, applications have control on the text conversion that are executed on a call by call basis.

Clipboard Bidirectional Attributes

The clipboard maintains two sets of bidirectional attributes.

The first set is called the 'Clipboard Bidirectional Attributes' and is associated with the text data that is currently stored in the clipboard. The default value for this set is inherited from the process bidirectional attributes of the application that issues WinSetClipbrdData. The source application can optionally modify this set of attributes by using the WinSetLangInfo API with the LI_BD_CLIP_ATTR effect.

The second set is called the 'Conversion Bidirectional Attributes'. This set is associated with the target application, and its default value is inherited from the process bidirectional attributes of the application that issues the WinQueryClipbrdData API. The target application can optionally modify these attributes by using the WinSetLangInfo API with LI_BD_CLIP_CONV_ATTR effect.

Refer to the Bidirectional Text Exchange Among Applications - Clipboard section for more details.

Managing Bidirectional Attributes Using WinSetLangInfo

The WinSetLangInfo and the WinQueryLangInfo APIs are used by applications to manage the various bidirectional attributes. The ulEffect parameter of these APIs determines what kind of bidirectional attribute is being set/queried by the call. The following values for ulEffect are supported:

- LI_BD_PROCESS_ATTR - Process bidirectional attributes.
- LI_BD_WND_ATTR - Window bidirectional attributes.
- LI_BD_CLIP_ATTR - Bidirectional attributes of the text contained in the clipboard.
- LI_BD_CLIP_CONV_ATTR - Bidirectional attributes to be used to convert the clipboard text data before being given to the application.
- LI_BD_WND_STAT - Window bidirectional status (see below).

For the LI_BD_WND_ATTR and LI_BD_WND_STAT effects, the caller of the API can choose if the setting of the bidi attributes (or status) should be done for the specified window itself, or for the specified window and all his child-windows. This is possible by specifying the LIF_CHILD_INHERIT flag bit in the flFlags parameter of the WinSetLangInfo API.

The caller can also instruct the window whose bidi attributes (or bidi status) are being set, to redraw itself, by specifying the LIF_WND_REFRESH flag.

Using WinSetLangInfo to Send WM_SETBIDIATTR Messages

As part of the WinSetLangInfo API, an optional message can be sent to the window procedure whose bidi attributes (or bidi status) is being set. This provides a bidi-aware window procedure with a way to know that an application is modifying its bidi state (attribute and/or status). This allows the window procedure to provide the required processing for this case. For example, when a bidi-aware window receives the WM_SETBIDIATTR message, it can inspect which bidi attributes are being set, and do necessary conversion on its internal buffers to accommodate this change. Another option it can take is to refuse the request and protect its bidi attributes from modifications done by external applications.

A window procedure can also ignore the WM_SETBIDIATTR by passing it to the default window procedure which will result in the bidi attributes being set in the internal window structure.

Similarly, the WinQueryLangInfo can send a message to the window procedure whose bidi attributes or status are being queried. This gives the bidi-aware window a chance to specify the value of the bidi attributes or status that is returned to the querying application.

A window procedure can also ignore the WM_QUERYBIDIATTR by passing it to the default window procedure which returns the window bidi attributes as stored in the internal window structure.

When the WinSetLangInfo API is used by a window procedure to set its own bidirectional attributes as part of WM_SETBIDIATTR processing, the LIF_NO_SENDMSG flag should be set in order to prevent recursive calls to the window procedure.

It is recommended that the LIF_NO_SENDMSG flag is NOT turned on when WinSet/QueryLanginfo is called outside the window procedure whose attributes are being set. Applications should generally use the default flag (which causes a WM_SETBIDIATTR message to be sent), so that bidi-aware window procedures can take the appropriate actions when a request to change their Bidi attributes is made.

Bidirectional Status

The following sections describe bidirectional status.

About Bidirectional Status

Every window in the system maintains its bidirectional status.

Bidirectional status flags are used to represent configuration information that tells the system how to handle various events for the window. The status flags include information that defines how bidirectional language hotkeys should be handled and its current typing mode.

The bidirectional status is saved by the system for each window and it can be set and queried by applications using the WinSetLangInfo API.

Managing Bidirectional Status Using WinSetLangInfo

The WinSetLangInfo API is used to manipulate the window bidi status word.

The Bidirectional status flags hold information about the following:

- Status of the bidirectional input support (enabled/disabled).
- Hotkey disable flags.
- AutoPush enable flags.

The PM Bidirectional Support defines several hotkeys that allow the user to perform Bidi-related operations such as Start Push or Field Reverse.

- Status of the bidirectional Input Support.

  When this flag is set ON, the bidirectional input support system is disabled. The input system operates in the same way as in a non-bidi (e.g US/English) system.

- Hotkey disable flags

  The hotkey disable flags are used by the system to process the various bidirectional hotkeys. When a certain flag is ON this tells the system that the default system action for this hotkey (identified by the flag) is disabled. Applications can disable hotkeys that use key combinations which the application wishes to handle itself.

  When a disabled bidirectional hotkey combination is detected, the system does not take any special action and passes the event to the application. This allows the application to receive notification about the hotkey being typed and take the necessary action. In addition, the system also generates a WM_CHAR event with a Virtual Key value (VK_) appropriate to this hotkey. This allows applications to isolate themselves from the actual key combination and be sensitive only to the virtual key value (this allows remapping of the bidi hotkeys).

  When a hotkey is enabled, the system will translate the key combination to the relevant bidi virtual key identifier. Applications that respond to the bidirectional hotkeys will take the appropriate action when they detect the virtual key identifiers. The system may also take some actions based on the hotkey being pressed (such as changing the keyboard layer, see below).

- AutoPush enable flags.

  Bidirectional status also contain enable/disable flags for 'Automatic Push'. These flags tell the system if the user has enabled the 'AutoPush' feature for the PM entry fields. (AutoPush enabled means that the system starts and ends Push mode automatically, based on the characters that are typed by the user. AutoPush is only relevant if the PM entry field contains visual text). There are two AutoPush flags, one for typing in Left-To-Right direction and one for typing in Right-To-Left.

Using WinSetLangInfo to Send WM_SETBIDISTAT Messages

The WM_SETBIDISTAT message is optionally sent to a window procedure every time a window bidirectional status word is set. Likewise, the WM_QUERYBIDISTAT can be sent when the status is queried. This lets a bidi-aware window procedure know that its status is being change and to take the appropriate action, if required. In general, these messages are sent and processed in the same way as the WM_SETBIDIATTR and WM_QUERYBIDIATTR message, as described in the "Sending WM_SETBIDIATTR messages in WinSetLangInfo API".

Keyboard Input and Layers

The following sections describe keyboard input and layers.

What Are Keyboard Layers?

On keyboards used for bidirectional languages, two characters (a Latin one and a "National Language" one) are engraved on each key and the user can select which of these characters is used, by selecting the active keyboard language (or "Keyboard Layer").

Changing the keyboard layer can be done by the user pressing one of the "Layer Change" hotkeys, or by an application issuing the WinSetKbdLayer API. In this way, the same physical keyboard is used to input text in both languages.

CUA guidelines define the "keyboard layer change" hotkeys as follows:

- Select Latin/English layer - Alt+Left Shift.
- Select National Language layer (Arabic/Hebrew) - Alt+Right Shift.

Managing Keyboard Layers Using WinSetKbdLayer

Keyboard layers are set and queried by the WinSetKbdLayer and the WinQueryKbdLayer APIs. These APIs are provided so that applications can set the keyboard to produce characters in the language that is required by the application for a specified window (or text input field). Issuing a WinSetKbdLayer API by an application is equivalent to pressing a "Layer Change" hotkey by the user.

WM_KBDLAYERCHANGED Message

When the keyboard layer is changed, a WM_KBDLAYERCHANGED is optionally posted to the window procedure that has keyboard focus, to notify it about the keyboard layer change. This lets the window procedure take the appropriate action (such as updating its internal variables and/or updating some status display).

The message is sent only when the WinSetKbdLayer API is specifically instructed to do so by specifying the SKLF_SENDMSG flag.

Bidirectional Hot Keys Processing

The system performs some processing for bidi hotkeys. When bidi hotkeys are detected, the system generates WM_CHAR messages containing the appropriate Bidi virtual keys (VK_XX values). The system may also take some default action depending on whether the specific hotkey is enabled or disabled (see below).

Note: The term Enable/Disable of Bidi Hotkey is a bit mis-leading. The effect of disabling a bidi hotkey does not neccessarily disable the hotkey itself - but the system default processing that is associated with this hotkey. For example, when the start push hotkey is disabled the Push Popup is NOT invoked (this is the default action taken by the system for VK_START_PUSH). The WM_CHAR/VK_START_PUSH message is generated in any case.

Whenever the system detects a Bidi hotkey which is enabled, the following actions take place:

- A WM_CHAR message containing the bidi VK value is generated.
- The message containing the hotkey combination is consumed.
- The system performs some default action for relevant hotkeys.

The following is the default action taken by the system for each enabled hot key:

- Set keyboard layer to Latin or National

  The appropriate keyboard layer is set for the window.
- Start push

  The Bidi "Push Popup" IME is invoked.
- End push

  No action is taken.
- Field reverse (toggle)

  Field reverse edit mode is activated.
- Window reverse (toggle)

  No action is taken.
- Activate auto push mode (toggle)

  No action is taken.
- Arabic shape selection hotkeys

  No action is taken.
- Activate Status Indicator

  The Language Viewer is invoked, if one is registered in the system.

Whenever the system detects a Bidi hotkey which is disabled, it does not generate the default action associated with this hotkey. Note that the generation of the VK_ value is NOT suppressed.

The following actions take place:

- A WM_CHAR message containing the bidi VK value is generated.
- The message containing the hotkey combination is not consumed. I.e. applications get both the VK value and the original key combination.
- No default action is performed by the system.

Bidi aware applications that wish to process some hotkeys themselves may choose to disable them in order to prevent the system from taking the default action.

For example, a wordprocessor would probably disable the VK_START_PUSH hotkey, to stop the system from bringing up the push popup dialog, since the application will provide right-to-left typing functionality itself.

Bidirectional Support in PM Controls

The following sections describe bidirectional support in PM controls.

About Bidirectional Support of PM Controls

PM Controls are windows which are provided as part of the PM system. PM controls are used by applications to carry out simple input and output tasks (such as simple data entry, and choice selection).

PM Controls are bidi-aware windows which support bidirectional attributes and status. Thus they are sensitive to the various settings in the bidi attributes and bidi status and take actions accordingly. For example, PM controls support the 'window orientation' bidi attribute by formatting and displaying themselves according to the current orientation setting. Another example is their awareness to the active keyboard layer, especially in windows that process text input, such as the single-line Entry Field (SLE).

By using the bidi-aware PM controls, applications can achieve high degree of bidirectional support, since many tasks that are specific to bidirectional support, are provided "automatically" by the controls, on behalf of the application. In other words, as PM controls relieve the application from implementing code for routine and simple tasks (such as text input), they also relieve the application (to an extent), from dealing with specific bidirectional processing required for these tasks.

Using BIDIPARAMs in Resource Scripts

PM Controls are usually defined in Resource Scripts. Usually, modfiying resources does not require the application developer to change code in the application.

The BIDIPARAM keyword is used to mark PM controls as having specific bidirectional features (attributes and status). Using the BIDIPARAM keyword in a resource script allows the application developer to change the definition of a PM control window, such that when created it behaves "automatically" in a certain bidirectional manner (e.g, it is formatted "automatically" as a "Right-To-Left" window, or is initially ready to accept Arabic/Hebrew text).

BIDIPARAMs are inherited to child windows. For example, specifying BIDIPARAMs for a dialog window resource, affects all child-windows in that dialog.

The BIDIPARAM keyword is specified with two parameters. The first one is the bidi feature that is manipulated (i.e a certain bidirectional attribute, or a certain bidirectional status flag) and the second parameter is the value that is given to that feature. In the following example, the dialog window is marked has having Right-To-Left window orientation. All child windows inherit this feature, except for the entry field which is specified has having a Left-To-Right window orientation. In addition the radio button is marked has having "Implicit" text type.

```
DLGTEMPLATE ID_SAMPLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
    DIALOG "Sample Bidirectional Dialog", ID_SAMPLE, 5, 10, 100, 50,
    FS_DLGBORDER
    BIDIPARAM      PP_BDATTR_WND_ORIENTATION  BDA_WND_ORIENT_RTL

    BEGIN

       ENTRYFIELD     "", ID_ENTRY,  5, 80, 40, 8
       BIDIPARAM      PP_BDATTR_WND_ORIENTATION BDA_WND_ORIENT_LTR

       RADIOBUTTON    "Radio", ID_RADIO, 5, 60, 40, 8
       BIDIPARAM      PP_BDATTR_TEXTTYPE BDA_TEXTTYPE_IMPLICIT
```

```
      CHECKBOX       "Check", ID_CHECK1, 5, 50, 40, 8
   END
END
```

The following is a list of PM controls with a short description of their bidirectional support behavior.

Button

The Button control handles its text according to its window bidi attributes.

Radio buttons and Checkboxes with right-to-left window orientation behave as follows:

- The Selection Icon (a little Box for Check Box, a little circle for Radio Buttons) is right justified. It is located in the right symmetric position, relative to its position in a &ltr Button.
- The Text of the Button, is displayed to the left of the Selection Icon, and is right justified.
- It switches the language layer to the national language , so it can process Arabic/Hebrew "Mnemonics" entered from the keyboard.

Combination-Box

The bidi attributes of the listbox part and the entryfield part of the combination box are "linked" (i.e changing one of them causes the other to change as well).

When the combination box has right-to-left window orientation the listbox is located beneath and to the left of the entryfield. The left side of the listbox is aligned with the left side of the entryfield.

Container

The Container control manages a hierarchy of bidi attributes which resembles the hierarchy of the container components.

A container control typically has the following hierarchy:

```
   Container --> Record(s) -->  Field(s)
```

Initially, records and fields do not have bidi attributes of their own (their bidi-attribute are not-initialized). At this time, their effective bidi attributes are those of the container window.

Applications can set their container's records or fields bidi attributes by sending the CM_SETITEMBIDIATTR and CM_SETFIELDBIDIATTR messages..

They can query their container's records and fields bidi attributes by sending CM_QUERYTEMBIDIATTR and CM_QUERYFIELDBIDIATTR.

The bidi-attributes hierarchy of the container works as follows:

- A field uses its own bidi attribute (set by CM_SETFIELDBIDIATTR).
- If its bidi-attribute has not been initalized (by sending the CM_SETFIELDBIDIATTR message), a field uses the bidi attribute of its record.
- A record uses it own bidi attribute (set by CM_SETITEMBIDIATTR).

- If its bidi-attribute has not been initialized (by sending the CM_SETITEMBIDIATTR), a field uses the bidi attribute of the container window.

The following is the description of the container window messages that are used to manage bidi attributes:

```
CM_SETITEMBIDIATTR - Set bidi attributes for a specific item (record)

   mp1 - PRECORDCORE pRecordCore
   mp2 - pointer to the following structure.

   typedef struct _BD_ATTR_MASK
         {
         ULONG  ulBD_Attr;
         ULONG  ulBD_Mask;
         } BD_ATTR_MASK;

CM_QUERYITEMBIDIATTR - Queries the bidi attributes of a specific
                       item (record).
   mp1 - PRECORDCORE pRecordCore
   mp2 - PULONG      pBidiAttr - points to a ULONG where BidiAttributes
                                 are returned.

CM_SETFIELDBIDIATTR: - Sets bidi attributes for a specific field (in
                       details view only).

   mp1 - PFIELDINFO pFieldInfo
   mp2 - pointer to the following structure.

   typedef struct _BD_ATTR_MASK
         {
         ULONG  ulBD_Attr;
         ULONG  ulBD_Mask;
         } BD_ATTR_MASK;

CM_QUERYFIELDBIDIATTR - Queries the bidi attributes of a specific field
                        (in details view only).

   mp1: PFIELDINFO pFieldInfo
   mp2: PULONG     pBidiAttr - points to a ULONG where BidiAttributes
                               are returned.
```

In general, when the window orientation of the Container is RtL its presentation and behavior is symmetric to the one used in LtR orientation. Some specific points are:

- "First" is considered on the right, "Next" is to the left. For example, the "First" view in a split details view is on the right and the "second" is on its left.
- Justification of fields is determined by the field attribute word (CFA_LEFT, CFA_RIGHT). If no justification is specified, then the default justification is determined according to the effective bidi attribute of the field (if WND_ORIENT_RTL then default justification is RIGHT). See more, below, on "effective" bidi attributes.
- Vertical scroll-bars are positioned at the left side of the container and horizontal scrollbars are assigned RtL orientation (see section on bidi support for scrollbars).

Dialog Box

The dialog box manager (which is activated using the WinDlgBox and WinProcessDlg APIs) inherits its OWNER window bidi attributes. This attributes are inherited to its child windows, except if the child window has predefined BIDIPARAM.

When the window orientation of the DialogBox is right to left, all child controls are positioned in their symmetric position in the window.

Entry-Field (Single-Line)

The single-line entry field processes its text according to its bidi attributes and according to the user interface convention used (Manual/Visual or Automatic/Implicit).

See more about the user interface of the entry field in the user documentation of the OS/2 Hebrew/Arabic National Language Version.

Entry Field (Multi-Line)

The multi-line entry field processes its text according to its bidi attributes and according to the user interface convention used (Manual/Visual or Automatic/Implicit).

See more about the user interface of the entry field in the &pmug. publication.

Frame

When the frame control has right-to-left window orientation, the position of its controls is symmetric to the position in left-to-right orientation, except for the system-menu, and min-max controls that stay at the same position.

The frame responds to the WM_SETBIDIATTR message by re-positioning of its controls. It also sends the message to the client window.

Listbox

The &rtl behavior of the List Box results in the justification of the text in each listbox item. In a &rtl Listbox, the text of the items is right-justified (as opposed to left-justification in the original, &ltr Listbox).

In addition, the display of the text is done in the correct order, according to the bidi attributes of the listbox window.

Menu

The menu control processes it items' text according to its bidi attributes.

When the menu's window orientation is RtL, the Menu control operates in a symmetric way relative to the behavior of the original (LtR) Menu control.

In RtL action Bars, the arrow keys behave in a visual way. I.e. The Right-Arrow key is considered to have a 'Backward' operation and the Left-Arrow key is considered to have 'Forward' operation.

The menu supports mnemonics in the "correct" keyboard layer. The menu control sets the keyboard layer based on the language of the first mnemonic character defined in the menu. This means, that it is expected that in a certain menu, the application uses only one language for mnemonic characters.

For Arabic: All character shapes of any Arabic mnemonic are accepted.

Notebook

Selecting various BKS_ notebook styles together with setting the window orientation to right-to-left can be used to construct a notebook

which resembles a right-to-left notebook.

When the notebook control has RtL window orientation, the page arrows have their roles reversed: the left arrow turns the pages forwards, whereas the right one - backwards. The text of Tabs, and Status line is converted according to their Bidi Attributes.

For Arabic, mnemonic specified in TABs are normalized to their base shapes.

The notebook control processes the views of its internal objects (Tabs, Status-Line and Page-Arrows) according to their bidi attributes. Applications are responsible for processing the views of their own page components (e.g dialog controls) by setting the appropriate bidi attributes to these components.

The internal objects use the bidi attributes of the notebook window, unless their bidi attributes are specifically set using one of the following messages:

- BKM_SETTABTEXTBIDIATTR
- BKM_SETSTATUSLINEBIDIATTR
- BKM_SETPAGEINFO

Applications can query the bidi attributes of their tabs and status-line page components using the following messages:

- BKM_QUERYTABTEXTBIDIATTR
- BKM_QUERYSTATUSLINEBIDIATTR
- BKM_QUERYPAGEINFO

When the notebook control receives the WM_SETBIDIATTR message it modifies its own bidi attributes (and as a result - its appearance).

Slider

The Value Set control processes it text according to its window bidi attributes.

The SLS_HOMERIGHT style of the Slider control is used to support RtL window orientation for this control.

Spin Button

The entryfield part of the Spin Button control inherits the same bidi attributes as those of the Spin Button window. It behaves in the same way as the 'normal' entry field.

When the SpinButton control has RtL window orientation, the location of the "arrow-window" is on the left side of the window.

Static

The Static control processes its text according to its window bidi attributes.

The DT_RIGHT flag is the default when the window has RtL orientation.

The DT_WORDBREAK is sensitive to the text orientation in the BiDi attributes word.

When SS_GROUPBOX is specified in Static controls that have RtL window orientation, the text is right-justified.

Title-Bar

The Title-Bar control processes its text according to its window bidi attributes.

When the Title-Bar control has Rtl window orientation:

- Text is right-justified.
- Text is truncated on the left side.

The same behavior (truncation on the left) is also performed if the titlebar text is all Arabic or Hebrew (i.e, does not contain any English character).

Value Set

The Value Set control processes it text according to its window bidi attributes.

The VS_RIGHTTOLEFT style is used to make the Value Set control have RtL window orientation.

Each item in the Value Set can have its own bidirectional attributes. Managing the item bidi attributes is performed by the following window messages:

- VM_SETITEMBIDIATTR - Sets a ValueSet Item's bidi attributes.
- VM_QUERYITEMBIDIATTR - Queries a ValueSet Item's bidi attributes.

If no item bidi attribute is set using the VM_SETITEMBIDIATTR message the item uses the window bidi attributes.

Parameters of VM_SETITEMBIDIATTR:

mp1 - MPFROM2SHORT(usLine, usColumn)

mp2 - pointer to the following structure.

```
typedef struct _BD_ATTR_MASK
      {
      ULONG  ulBD_Attr;
      ULONG  ulBD_Mask;
      } BD_ATTR_MASK
```

Parameters of VM_QUERYITEMBIDIATTR:

mp1 - MPFROM2SHORT(usLine, usColumn)

mp2 - PULONG pBidiAttr - points to a ULONG where bidi attributes are returned.

Bidirectional Support in Standard Dialogs

All standard dialogs (File dialog, Font dialog and the Messagebox) support a Right-To-Left/NLS mode. In this mode, their standard text is in the national language and the general orientation of the dialog is right-to-left. Standard dialogs can be made to work in this mode as a result of an application interface (specific for each standard dialog) or when the the environment variable STDDLGLANG is set to 'NATIONAL'. The following is a description of the Right-To-Left/NLS mode of each of the standard dialogs:

File Dialog

Application interface: The FDS_NATIONAL_LANGUAGE 'style' bit.

When the standard file dialog is RtL/NLS:

- Orientation is right-to-left.
- Field names and buttons are in the National Language.
- Data in controls that pertain to filenames and directory names is implicit.

Font Dialog

Application interface: The FNTS_NATIONAL_LANGUAGE 'style' bit.

When the standard font dialog is RtL/NLS:

- Orientation is right-to-left.
- Field names and buttons are in the National Language.
- Data in controls that pertain to font names is VISUAL for Hebrew and Implicit for Arabic.

Message Box

Application interface: Owner Window orientation is right-to-left.

When the message box is RtL/NLS:

- Text is right-justified in the titlebar and inside the messagebox.
- Icons (if specified) are on the right side of the messagebox.
- Push buttons are ordered from right-to-left.
- Push buttons have their text in the national language.

Bidirectional Text Exchange among Applications

The following sections describe bidirectional text exchange among applications.

Clipboard

The clipboard is a mechanism provided by PM to allow the user to transfer data between different applications. To do so, the user selects some data in a window, "copies" (or "cuts") it to the clipboard and then "pastes" the data in the target application.

When dealing with bidirectional data, it may be required to transfer text data, that has different bidirectional attributes (i.e, different format of text) between applications. Accordingly, a conversion of the text must be performed, in order to transform the text to a format (identified by bidirectional attributes) which is compatible with the target application (the one that "pastes" the data).

The PM bidirectional support provides automatic conversion of text between source and target applications, based on their process

bidirectional attributes. In addition, bidi-aware applications can explicitly set the bidirectional attributes managed by the clipboard in order to affect the conversion that is performed on the text.

In order to allow the automatic conversion of the clipboard, the following conditions must be met:

- Both the source and the target applications have valid process bidirectional attributes (i.e, their bidi attributes are not 0).
- Exchange of CF_TEXT and CF_DSPTEXT format is performed.
- The source and target bidi attributes are different.
- None of the applications (i.e, neither the source application nor the target one) have issued any APIs to cancel the conversion provided by the clipboard.

The clipboard maintains two sets of bidirectional attributes.

The first set is called the 'Clipboard Bidirectional Attributes' and is associated with the text data that is currently stored in the clipboard. The default value for this set is inherited from the process bidirectional attributes of the application that issues WinSetClipbrdData. The source application can optionally modify this set of attributes by using the WinSetLangInfo API with the LI_BD_CLIP_ATTR effect.

The second set is called the 'Conversion Bidirectional Attributes'. This set is associated with the target application, and its default value is inherited from the process bidirectional attributes of the application that issues the WinQueryClipbrdData API. The target application can optionally modify these attributes by using the WinSetLangInfo API with LI_BD_CLIP_CONV_ATTR effect.

When using the WinSetLangInfo API to change the clipboard data, bidi-aware applications must follow the following rules:

- Setting of both the 'Clipboard bidi attributes' and the 'conversion bidi attributes' must be performed only when an application has access to the clipboard - i.e, after issuing WinOpenClipbrd and before issuing the WinCloseClipbrd APIs.
- Setting LI_BD_CLIP_ATTR must be done AFTER data has been put into the clipboard (that is, after issuing WinSetClipbrdData).

The following is a typical scenario. Lines marked with '(BD)' are optional, and can be issued by bidi-aware applications.

```
Setting Clipboard Bidirectional Data.
-------------------------------------

WinOpenClipbrd()
WinEmptyClipbrd()
WinSetClipbrdData(CF_TEXT)     (clipboard bidi attributes are set -
                                inheited from process bidi attributes).

(BD) WinSetLangInfo(LI_BD_CLIP_ATTR..) (clipboard bidi attribute are set)

WinCloseClipbrd()


Querying Clipboard Bidirectional Data
-------------------------------------

 WinOpenClipbrd()

 (BD) WinQueryLangInfo(LI_BD_CLIP_ATTR...)
                                (Queries the current clipboard
                                 bidi attributes).
 (BD) WinSetLangInfo(LI_BD_CLIP_CONV_ATTR...)
                                (sets the clipboard conversion
                                 bidi attributes).
  WinQueryClipbrdData(CF_TEXT)   (clipboard performs conversion on
                                  the text, using the clipboard
                                  and the conversion bidi attributes.
  WinCloseClipbrd()
```

Dynamic Data Exchange (DDE)

Dynamic data exchange is performed between windows. Thus, conversion of bidirectional text, if required, is based on the bidirectional attributes of the windows which are involved in the data exchange.

- The conversion is performed at WinDdePostMsg() time, using the hwndTo and hwndFrom bidi attributes (which are parameters of the WinDdePostMsg API).
- Bidi-aware applications can force no conversion by setting the bidi attributes of any window that is involved in the data exchange to 0.
- Conversion is done only for WM_DDE_DATA messages and only when the format of data being exchanged is DDEFMT_TEXT.

Notes:

- It is recommended that application names and item names (that are used by an application in the process of "making the DDE contact" with other applications) are in English. This is because conversions occur only at WM_DDE_DATA and not in WM_DDE_INITIATE time, and also because at WM_DDE_INITIATE time, there is no window handle from which the system can deduce the bidi attribute of the application name and item text.
- It is recommended that bidi-aware applications use "invisible" windows (or object windows) as the DDE source and target windows. This gives them the freedom to set their windows bidi attributes to any value without having to worry about the visual effect this may have on the display.

Direct Manipulation

There is no specific bidirectional support implemented in the system support for Direct Manipulation.

Bidi-Aware applications that add their own type of 'direct manipulation mechanism' (such as a spreadsheet program externalizing its drag & drop mechanism, so that another application can interface with it), must publish, as part of their interface specification, the bidi attributes, so that other applications, which may use different types of text, can do the necessary bidi text conversion, if required.

Information Presentation Facility

The Bidirectional support in the PM Information Presentation Facility has the following components:

- A modified version of the Help Compiler named IPFCBIDI. This component provides all the functions of the regular IPFC program, with the addition of some Bidirectional features.
- Runtime support for Bidirectional help documents.

Bidirectional Features of the IPFCBIDI Compiler

The following sections describe the bidirectional features of the IPFCBIDI compiler.

Compiler Invocation

IPFCBIDI accepts new values for the "/LANGUAGE" parameter:

ARA	for Arabic
HEB	for Hebrew

Examples using the new values:

```
IPFCBIDI myfile.txt /INF /COUNTRY=785 /CODEPAGE=864 /LANGUAGE=ARA
IPFCBIDI myfile.txt /INF /COUNTRY=972 /CODEPAGE=862 /LANGUAGE=HEB
```

Note that the Bidirectional functionality of the compiler and of the Help Viewer is only enabled when the language is explicitly specified as Arabic or Hebrew.

Grammar Files

When the new values of "/Language" are used, the corresponding names for the grammar files are:

- IPFARA.NLS
- IPFHEB.NLS.

When the language is Arabic or Hebrew, it is expected that at least one statement in the corresponding grammar file will be affected: the "WORD" statement should define both Latin characters and National (Arabic or Hebrew) letters.

Bidirectional Source File Format

When the language is Arabic or Hebrew, the .IPF source file must have one of the following formats:

Implicit
: Latin and National (Arabic or Hebrew) characters are arranged in typing order in successive positions of the source file.

  Arabic characters should be stored in their base shapes.

Visual - left to right
: Characters are stored in an order ready for display or printing

Visual - right to left
: The order of the characters is the mirror image of the previous format.

It is possible to specify the format of the source file by adding a special attribute to the "USERDOC" tag, as in the following examples:

```
:userdoc text=IMPLICIT.
:userdoc text=VISUAL_LTR.
:userdoc text=VISUAL_RTL.
```

If the text attribute is not specified with the "USERDOC" tag, the default is:

```
:userdoc text=IMPLICIT.
```

Another attribute that can be added to the "USERDOC" tag is **SWAP**. This attribute defines the way symmetric characters (such as '(' and ')', or '<' and '>') are stored in the .IPF file within right-to-left text.

The valid values for the SWAP attribute are:

ON
: This indicates that the .IPF source file has been created with symmetric swapping. For example, the 'Open parentheses' character within a right-to-left text is stored as '('. This is the default value for this attribute.

OFF
This indicates that the .IPF source file has been created without symmetric swapping. For example, the 'Open parentheses' character within a right-to-left text is stored as ')'.

Example:

```
:userdoc text=IMPLICIT swap=OFF.
```

Justification

When language is Arabic or Hebrew, the default justification is changed to right justification for the following:

1. line alignment ("align" attribute of the "LINES" tag)
2. figure alignment ("align" attribute of the "FIG" tag)
3. bitmap alignment ("align" attribute of the "ARTWORK" tag)
4. window positioning ("x" attribute of the "H" tag) of a window relatively to its parent window.

Examples:

The tag below causes a window to be right-justified:

```
:h1 id=sample1. This is a right-justified Window.
```

The tag below causes a window to be left-justified:

```
:h1 id=sample21 x=left. This is a left-justified Window.
```

The tag below causes the next lines to be left aligned:

```
:lines align=left.
```

The tag below causes the next lines to be right aligned:

```
 :lines.
```

Runtime Support for Bidirectional Help Documents

The runtime support for Bidirectional help documents (both .inf and .hlp files) is active only for documents that were compiled with IPFCBIDI using /L=ARA or /L=HEB. Such documents will be referred to as "Bidi documents."

Documents compiled with the non-bidi ipfc compiler and documents compiled with IPFCBIDI and any non-bidi language will not be affected by the Bidi runtime support.

User Interface

When a Bidi document is loaded, the user interface of the help manager, including menus, dialogs, buttons and help will be in the national language.

Display

The reading direction for Bidi documents is from right to left. Help manager structures in general will be displayed as a mirror image of their non-Bidi counterparts. This means they will be right justified instead of left justified.

Some examples of such help manager structures include:

- Headers
- Table of contents
- Figures
- Footnotes
- Index
- Paragraphs
- Lines
- Lists of all kinds
- Examples
- Tables

  Columns within a table go logically from right to left.

Concatenation of INF Files

When multiple inf files are viewed together, the helpmanager will behave as if the entire document is Bidi if any one of the inf files was compiled as a Bidi document.

The Language Viewer

The following sections describe the language viewer.

About the Language Viewer

The Language Viewer is a tool which is provided to help the user identify the current state of the language support, by providing visual indications that represent the current bidirectional state of the window that has the focus.

The default Viewer is implemented as a PM/WorkplaceShell application that shows a number of icons that represents the current bidirectional attributes as well as the active keyboard layer of the window that has the PM focus.

The Viewer is optional. The PM bidirectional support works even if the viewer application is not active. In addition, the viewer can be replaced by a user-written application. Such an application can modify the visual effect that is provided to the user, allowing for more flexibility of the user-interface.

Setting a New Language Viewer

An application that wishes to act as a Language Viewer uses the WinSetLangViewer API. This allows the application to define a window handle that is known to PM to be the new Language Viewer window. PM supports only one Language Viewer. That is, when WinSetLangViewer is issued, the window handle that is referenced becomes the new viewer, and the old viewer does not function as a viewer anymore.

When a window acts as a Language Viewer it receives WM_LANGVIEWINFOCHANGED messages from the system. These messages define the current changes in bidirectional state that are made to the window that has the focus. Also, PM posts a WM_LANGVIEWINFOCHANGED message to the Language Viewer window to notify it that it should update its display, since a new window (probably with a different set of bidi attributes and keyboard layer) becomes the focus window.

Invoking the Options Dialog

The default Viewer invokes the option dialog in response to the WM_LANGOPTIONSDIALOG message. This is sent to the viewer by the system in response to the "Invoke language viewer" hotkey, but may also be sent to the viewer by any application.

Environment Variables Used for Language Support

The OS/2 Language support for bidirectional languages uses environment variables as a means of configuring specific applications. Setting environment variables is similar to setting the language settings in the Program and ProgramFile Workplace Shell objects.

The following environment variables are supported:

- BIDIATTR - Defines the Bidirectional Attributes for the session.

  Possible values for this keyword:

  - INIT - Initializes a Bidi Attribute.
  - TEXTTYPE_VISUAL - Visual type of text.
  - TEXTTYPE_IMPLICIT - Implicit type of text.
  - WND_ORIENT_LTR - Left-to-right window orientation
  - WND_ORIENT_RTL - Right-to-left window orientation
  - TEXT_ORIENT_LTR - Left-to-right text orientation
  - TEXT_ORIENT_RTL - Right-to-left text orientation
  - TEXT_ORIENT_CONTEXT - Text orientation is determined dynamically upon the context (based on the actual characters that construct the text string).
  - SYM_SWAP_OFF - Symmetric swapping of directional characters is not handled automatically.
  - SYM_SWAP_ON - Symmetric swapping of directional characters is handled automatically.
  - NUMERALS_NOMINAL - Display only nominal (i.e. Arabic) numerals
  - NUMERALS_NATIONAL - Display only national (i.e. Hindi) numerals
  - NUMERALS_CONTEXT - Display numerals according to surrounding text
  - NUMERALS_PASSTHRU - Do not process numerals on display
  - TEXT_DISPLAY_SHAPED - Automatically shape text upon display

    The following are valid shaping modes, but it is not recommended to use them as system settings in CONFIG.SYS.
  - TEXT_SAVE_SHAPED - Assume that text is already shaped, and automatically shape new text as it is being entered by the user.
  - TEXT_SHAPE_INITIAL - Assume that text is already shaped and new text is entered in initial mode
  - TEXT_SHAPE_MIDDLE - Assume that text is already shaped and new text is entered in middle mode
  - TEXT_SHAPE_FINAL - Assume that text is already shaped and new text is entered in final mode
  - TEXT_SHAPE_ISOLATED - Assume that text is already shaped and new text is entered in isolated mode
  - TEXT_SHAPE_NOMINAL - Display and store text in its nominal (base) shapes.

  Example: SET BIDIATTR=WND_ORIENT_RTL, TEXTTYPE_IMPLICIT

- BIDISTAT - determines the configuration of the system support for bidirectional languages. Allowed values are:

  - DISABLE_INPUT_PROCESSING - disables all processing done by the system for input (all hotkeys are disabled).
  - HKFLAG_ENG_LAYER - disables the "English Language" hotkey.
  - HKFLAG_NAT_LAYER - disables the "National Language" hotkey.
  - HKFLAG_PUSH - disables the "Start Push" hotkey.

- HKFLAG_END_PUSH - disables the "End Push" hotkey.
- HKFLAG_AUTO_PUSH - disables the "AutoPush" toggle hotkey.
- HKFLAG_FIELD_REV - disables the "Field Reverse" hotkey.
- HKFLAG_SCREEN_REV - disables the "Window Reverse" hotkey.
- HKFLAG_STATUS_INDICATOR - disables the "Language Viewer" hotkey.
- HKFLAG_DISPLAY_SHAPED - disables the "Display Shaped/Automatic" hotkey.
- HKFLAG_INITIAL - disables the "Initial" hotkey
- HKFLAG_MIDDLE - disables the "Middle" hotkey
- HKFLAG_FINAL - disables the "Final" hotkey
- HKFLAG_ISOLATED - disables the "Isolated" hotkey
- HKFLAG_SAVE_SHAPED - disables the "Save shaped" hotkey
- FAUTOPUSH_RTL_ON - Sets AutoPush for right-to-left fields ON.
- FAUTOPUSH_LTR_ON - Sets AutoPush for left-to-right fields ON.

Example: SET BIDISTAT=HKFLAG_STATUS_INDICATOR

- STDDLGLANG - configures the language and appearance of standard dialog (like "Save file" dialog) and message box buttons. Allowed values are:

  - ENGLISH - Language is English and orientation is left-to-right.
  - NATIONAL - Language is National (Hebrew/Arabic) and orientation is right-to-left.

  Example: SET STDDLGLANG=NATIONAL

- BIDIUI (For Hebrew ONLY). - configures the default text entry user interface for the session.

  Allowed values are:

  - VISUAL - Default text entry interface is Visual.
  - IMPLICIT - Default text entry interface is Implicit.
  - CONTEXT - Default text entry interface is defined by Context (i.e, defined by the type of text of the etnry field - Entry fields with Visual text type will have Visual user interface by default and entry fields with Implicit text type will have Implicit user interface by default).

  Example: SET BIDIUI=VISUAL

Building Bidirectional Programs

The following sections tell you how to build bidirectional programs.

Include Files

To be able to use the bidirectional support two include files should be added to your INCLUDE path. They are:

- layout.h - This file defines the basic structures as well as the bidi text conversion APIs.
- pmbidi.h - This file is PM-related, and it contains all the definitions and API prototypes for writing a PM bidirectional program.

How to Use and Link to the Bidirectional Functions

All the bidirectional functions are contained in the executable file PMBIDI.DLL. In order to dynamically link your application with PMBIDI.DLL, you have to use the import library PMBIDI.LIB, which is included in this toolkit.

Sample Applications

Several sample applications are provided with source to demonstrate the use of the bidirectional functions.

Style

This sample program demonstrates the use of the Bidi APIs from a Bidi-Aware application.

It demonstrates the main techniques that are used by a bidi-aware PM application and integrates the information that is included in this guide.

To start the sample program do the following:

- Read the README file for a description of the bidi "tasks" that are handled by this sample.
- Run RUNSAMP.CMD or RUNSAMP1.CMD to activate the sample program.

Telephone Directory

This is a simple telephone directory application that demonstrates how to use the bidirectional functions to create applications with a bilingual (National/English) user interface.

To activate the sample run 'TELDIR'.

Data Types

This section describes the data types that are used in the Bidi functions. These are in the C language. For a description of a particular data type, select the + on the Contents to view a list of data types, then select the data type from that list.

Implicit Pointer Data Types

Data type names beginning with "P" (for example, PERRORCODE) is likely to be a pointer to another data type (in this instance, ERRORCODE).

In the data type summary, (provided by selecting the + in front of Data Types on the Contents), no explicit "typedefs" are shown for pointers. Therefore, if no data type definition can be found in the summary for a data type name "Pxxxxxx", it becomes a pointer to the data type "xxxxxx", for which a definition should be found in the summary.

The implicit type definition needed for such a pointer "Pxxxxxx" is:

```
typedef xxxxxx *Pxxxxxx;
```

Such definitions are provided by means of the system header file OS2.H.

Storage Mapping of Data Types

The storage mapping of the data types is dependent on the machine architecture. To be portable, applications must access the data types using the definitions supplied for that environment.

BD_ATTR_MASK

BD_ATTR_MASK Bidi attribute mask structure.

```
typedef struct  _BD_ATTR_MASK {
ULONG    ulBd_Attr;       /* Bidi attributes */
ULONG    ulBd_Mask;       /* Bidi attribute mask */
 } BD_ATTR_MASK;
```

---

ulBD_Attr

ulBD_Attr (ULONG)
Bidi attributes.

Contains the BIDIATTR values.

---

ulBD_Mask

ulBD_Mask (ULONG)
Bidi attribute mask.

Contains the BIDIATTRM values.

BDA_TEXT_ORIENT_CONTEXT

```
#define BDA_TEXT_ORIENT_CONTEXT     0x00020000UL
```

BDA_LEVEL

```
#define BDA_LEVEL                   0x30000000UL
```

BDA_INIT

```
#define BDA_INIT                    0x80000000UL
```

BDA_DATATYPE_VISUAL

```
#define BDA_DATATYPE_VISUAL         0x00000000UL
```

BDA_DATATYPE_IMPLICIT

```
#define BDA_DATATYPE_IMPLICIT       0x01000000UL
```

BDA_TEXT_ORIENT_LTR

```
#define BDA_TEXT_ORIENT_LTR         0x00000000UL
```

BDA_TEXT_ORIENT_RTL

```
#define BDA_TEXT_ORIENT_RTL         0x00010000UL
```

BDA_TEXT_ORIENT_CONTEXT

```
#define BDA_TEXT_ORIENT_CONTEXT     0x00020000UL
```

BDA_WND_ORIENT_LTR

```
#define BDA_WND_ORIENT_LTR         0x00000000UL
```

BDA_WND_ORIENT_RTL

```
#define BDA_WND_ORIENT_RTL         0x00100000UL
```

BDA_NUMERALS_ARABIC

```
#define BDA_NUMERALS_ARABIC        0x00000000UL
```

BDA_NUMERALS_PASSTHRU

```
#define BDA_NUMERALS_PASSTHRU      0x00001000UL
```

BDA_NUMERALS_HINDI

```
#define BDA_NUMERALS_HINDI         0x00002000UL
```

BDA_NUMERALS_CONTEXT

```
#define BDA_NUMERALS_CONTEXT       0x00003000UL
```

BDA_WORDBREAK_OFF

```
#define BDA_WORDBREAK_OFF          0x00000000UL
```

BDA_WORDBREAK_ON

```
#define BDA_WORDBREAK_ON           0x00000200UL
```

BDA_SYM_SWAP_OFF

```
#define BDA_SYM_SWAP_OFF           0x00000000UL
```

BDA_SYM_SWAP_ON

```
#define BDA_SYM_SWAP_ON            0x00000100UL
```

BDA_CSD_ON

```
#define BDA_CSD_ON                 0x00000000UL
```

BDA_CSD_PASSTHRU

```
#define BDA_CSD_PASSTHRU           0x00000001UL
```

BDA_CSD_BASE

```
#define BDA_CSD_BASE               0x00000010UL
```

BDA_CSD_INITIAL

```
#define BDA_CSD_INITIAL            0x00000011UL
```

BDA_CSD_MIDDLE

```
#define BDA_CSD_MIDDLE             0x00000012UL
```

BDA_CSD_FINAL

```
#define BDA_CSD_FINAL              0x00000013UL
```

BDA_CSD_ISOLATED

```
#define BDA_CSD_ISOLATED           0x00000014UL
```

BIDIATTR

The BIDIATTR is a 32-bit word that determines all the aspects of Bidi attributes related processing and window display associated with a specific application. The user controls the behavior of different applications (having different types of Bidi text) by manipulating these.

The following is the Bidi attributes structure:

BYTE	BIT	EXPLANATION	VALUE
BYTE 3	Bit 7	Initialized	BDA_INIT (PM ONLY)
	Bit 4-6	level	BDA_LEVEL
	Bit 0-3	Type of support	BDA_DATATYPE_VISUAL BDA_DATATYPE_IMPLICIT
BYTE 2	Bit 0-1	Orientation(text)	BDA_TEXT_ORIENT_LTR BDA_TEXT_ORIENT_RTL BDA_TEXT_ORIENT_CONTEXT (PM ONLY)
	Bit 2-3	Reserved	(must be zero)
	Bit 4	Window Orientation	BDA_WND_ORIENT_LTR BDA_WND_ORIENT_RTL
	Bit 5-7	Reserved	(must be zero)
BYTE 1	Bit 4-7	Numeral Shapes	BDA_NUMERALS_ARABIC BDA_NUMERALS_PASSTHRU BDA_NUMERALS_HINDI BDA_NUMERALS_CONTEXT (PM ONLY)
	Bit 2-3	Reserved	(must be zero)
	Bit 1	Word Break mode	BDA_WORDBREAK_OFF (PM ONLY) BDA_WORDBREAK_ON (PM ONLY)
	Bit 0	Symmetric Swapping	BDA_SYM_SWAP_OFF (PM ONLY) BDA_SYM_SWAP_ON (PM ONLY)
BYTE 0	Character Shape Determination		BDA_CSD_ON BDA_CSD_PASSTHRU BDA_CSD_BASE BDA_CSD_INITIAL BDA_CSD_MIDDLE BDA_CSD_FINAL BDA_CSD_ISOLATED

BIDIATTRM

The BIDIATTRM contains a value that determines which attribute(s) will be changed and which ones are masked.

The following are possible Bidi attribute mask values:

```
#define BDAM_INIT               0x80000000UL
#define BDAM_LEVEL              0x70000000UL
#define BDAM_WORD_BREAK         0x00000200UL
#define BDAM_DATATYPE           0x01000000UL
#define BDAM_TEXT_ORIENTATION   0x00030000UL
#define BDAM_WND_ORIENTATION    0x00100000UL
#define BDAM_NUMERALS           0x00003000UL
```

```
#define BDAM_SYM_SWAP           0x00000100UL
#define BDAM_CSD                0x000000FFUL

#define BDAM_ALL               (BDAM_DATATYPE         |
                                BDAM_TEXT_ORIENTATION |
                                BDAM_WND_ORIENTATION  |
                                BDAM_NUMERALS         |
                                BDAM_SYM_SWAP         |
                                BDAM_CSD               )
```

BIDIPARAM

The BIDIPARAM keyword can be used to initially set the Bidirectional attributes (see BIDIATTR) during window creation time. The usage of BIDIPARAM is analogous to that of the PRESPARAMS keyword.

Values defined by BIDIPARAM will override any inherited attributes at window creation time. However, it is important to note that the BIDIPARAM cannot be used to dynamicaly set the Bidirectional attributes after creation. Dynamic setting of the Bidirectional should be done using the PM Bidirectional window functions.

```
#define PP_BDATTR_FIRST            0x100L  /* First BidiAttr PresParam  */
                                           /*                           */
#define PP_BDATTR_ALL              0x101L  /* Set ALL Bidi attrs        */
                                           /*                           */
#define PP_BDATTR_TEXTTYPE         0x102L  /* Text/Data type            */
#define PP_BDATTR_TEXT_ORIENTATION 0x103L  /* Text Orientation          */
#define PP_BDATTR_WND_ORIENTATION  0x104L  /* Window Orientation        */
#define PP_BDATTR_NUMERALS         0x105L  /* Arabic/Hindi Numerals     */
#define PP_BDATTR_SYM_SWAP         0x106L  /* Symetric Swapping         */
#define PP_BDATTR_WORD_BREAK       0x107L  /* Word break                */
#define PP_BDATTR_TEXT_SHAPE       0x108L  /* Text Shape Determination  */
                                           /*                           */
#define PP_BDATTR_LAST             0x108L  /* Last BidiAttr PresParam   */
                                           /*                           */
#define PP_BDSTATUS                0x110L  /* Bidirectional status flags*/
```

BDS_FAUTOPUSH_RTL_ON

```
#define BDS_FAUTOPUSH_RTL_ON            0x00000001UL
```

BDS_FAUTOPUSH_LTR_ON

```
#define BDS_FAUTOPUSH_LTR_ON            0x00000002UL
```

BDS_FPUSH_ON

```
#define BDS_FPUSH_ON                    0x00000004UL
```

BDS_FKBD_LAYER_IS_NL

```
#define BDS_FKBD_LAYER_IS_NL            0x00000010UL
```

BDS_HKFLAG_FIELD_REV

```
#define BDS_HKFLAG_FIELD_REV            0x00200000UL
```

BDS_HKFLAG_ENG_LAYER

```
#define BDS_HKFLAG_ENG_LAYER            0x00010000UL
```

BDS_HKFLAG_NAT_LAYER

```
#define BDS_HKFLAG_NAT_LAYER            0x00020000UL
```

BDS_HKFLAG_PUSH

```
#define BDS_HKFLAG_PUSH                 0x00040000UL
```

BDS_HKFLAG_END_PUSH

```
#define BDS_HKFLAG_END_PUSH             0x00080000UL
```

BDS_HKFLAG_AUTO_PUSH

```
#define BDS_HKFLAG_AUTO_PUSH            0x00100000UL
```

BDS_HKFLAG_SCREEN_REV

```
#define BDS_HKFLAG_SCREEN_REV           0x00400000UL
```

BDS_HKFLAG_BIDI_POPUP

```
#define BDS_HKFLAG_BIDI_POPUP           0x02000000UL
```

BDS_HKFLAG_AUTOMATIC

```
#define BDS_HKFLAG_AUTOMATIC            0x04000000UL
```

BDS_HKFLAG_INITIAL

```
#define BDS_HKFLAG_INITIAL              0x08000000UL
```

BDS_HKFLAG_MIDDLE

```
#define BDS_HKFLAG_MIDDLE               0x10000000UL
```

BDS_HKFLAG_FINAL

```
#define BDS_HKFLAG_FINAL                0x20000000UL
```

BDS_HKFLAG_ISOLATED

```
#define BDS_HKFLAG_ISOLATED             0x40000000UL
```

BDS_HKFLAG_ISOLATED

```
#define BDS_HKFLAG_PASSTHRU             0x80000000UL
```

BDS_HKFLAG_ISOLATED

```
#define BDS_DISABLE_INPUT_PROCESSING    0x00000020UL
```

BIDISTAT

The BIDISTAT is a 32-bit word that determines all the aspects of Bidi status related processing and window display associated with a specific application. The user controls the behavior of different applications (having different types of Bidi text) by manipulating these.

The following is the Bidi status structure:

```
Bits 16-31 : Hot Key Enable/Disable Flags

Bit 31          Passthru shape hot key                  BDS_HKFLAG_PASSTHRU
Bit 30          Isolated shape hot key                  BDS_HKFLAG_ISOLATED
Bit 29          Final shapes hot key                    BDS_HKFLAG_FINAL
Bit 28          Middle shapes hot key                   BDS_HKFLAG_MIDDLE
Bit 27          Initial shapes hot key                  BDS_HKFLAG_INITIAL
Bit 26          Automatic/base shapes hot key           BDS_HKFLAG_AUTOMATIC
Bit 25          Bidi viewer invocation hot key          BDS_HKFLAG_BIDI_POPUP
Bit 24          RESERVED                                (must be zero)
Bit 23          RESERVED                                (must be zero)
Bit 22          Window reverse hot key                  BDS_HKFLAG_SCREEN_REV
Bit 21          RESERVED                                (must be zero)
Bit 20          Auto push hot key                       BDS_HKFLAG_AUTO_PUSH
Bit 19          End  push hot key                       BDS_HKFLAG_END_PUSH
Bit 18          Start push hot key                      BDS_HKFLAG_PUSH
Bit 17          Keyboard National layer hot key         BDS_HKFLAG_NAT_LAYER
Bit 16          Keyboard English layer hot key          BDS_HKFLAG_ENG_LAYER

Bits 0-15 : Bidi Flags

Bit 8-15        Reserved                                (must be zero)
Bit 7           Refresh flag (refresh PVB in full screen) BDS_HKFLAG_FIELD_REV
Bit 6           Reserved                                (must be zero)
Bit 5           Disable input processing                BDS_DISABLE_INPUT_PROCESSING
Bit 4           Reserved                                (must be zero)
Bit 2-3         Push state ON/OFF                       BDS_FPUSH_ON
Bit 1           Autopush ON/OFF in LtR window           BDS_FAUTOPUSH_LTR_ON
Bit 0           Autopush ON/OFF in RtL window           BDS_FAUTOPUSH_RTL_ON
```

BIDISTATM

The BIDISTATM contains a value that determines which status will be changed and which ones are masked.

The following are possible Bidi status mask values:

```
#define BDSM_HKFLAGS                    0x7E7F0000UL
#define BDSM_AUTOPUSH_RTL               0x00000001UL
#define BDSM_AUTOPUSH_LTR               0x00000002UL
#define BDSM_PUSH_ON                    0x00000004UL
#define BDSM_DISABLE_INPUT_PROCESSING   0x00000020UL

#define BDSM_ALL                      ( BDSM_HKFLAGS        |
                                        BDSM_AUTOPUSH_RTL   |
                                        BDSM_AUTOPUSH_LTR   |
                                        BDSM_PUSH_ON        |
                                        BDSM_DISABLE_INPUT_PROCESSING )
```

BOOL

BOOL Boolean.

Valid values are FALSE, which is 0, and TRUE, which is 1.

```
typedef unsigned long BOOL;
```

CDATE

CDATE Structure that contains date information for a data element in the details view of a container control.

```
typedef struct _CDATE {
UCHAR     day;    /* Day  */
UCHAR     month;  /* Month  */
USHORT    year;   /* Year  */
 } CDATE;
```

---

day

day (UCHAR)
Day.

---

month

month (UCHAR)
Month.

---

year

year (USHORT)
Year.

CHAR

CHAR Single-byte character.

```
#define CHAR char
```

CNRINFO

CNRINFO Structure that contains information about the container.

```
typedef struct _CNRINFO {
ULONG         cb;                 /* Structure size  */
PVOID         pSortRecord;        /* Pointer or NULL  */
PFIELDINFO    pFieldInfoLast;     /* Pointer or NULL  */
PFIELDINFO    pFieldInfoObject;   /* Pointer  */
PSZ           pszCnrTitle;        /* Title text or NULL  */
ULONG         flWindowAttr;       /* Window attributes  */
POINTL        ptlOrigin;          /* Workspace origin  */
ULONG         cDelta;             /* Threshold  */
ULONG         cRecords;           /* Number of records  */
SIZEL         slBitmapOrIcon;     /* Icon/bit-map size  */
SIZEL         slTreeBitmapOrIcon; /* Icon/bit-map size  */
HBITMAP       hbmExpanded;        /* Bit-map handle  */
HBITMAP       hbmCollapsed;       /* Bit-map handle  */
HPOINTER      hptrExpanded;       /* Icon handle  */
HPOINTER      hptrCollapsed;      /* Icon handle  */
LONG          cyLineSpacing;      /* Vertical space  */
LONG          cxTreeIndent;       /* Horizontal space  */
LONG          cxTreeLine;         /* Line width  */
ULONG         cFields;            /* Number of columns  */
LONG          xVertSplitbar;      /* Split bar position  */
 } CNRINFO;
```

---

cb

cb (ULONG)
Structure size.

The size (in bytes) of the CNRINFO data structure.

---

pSortRecord

pSortRecord (PVOID)
Pointer or NULL.

Pointer to the comparison function for sorting container records. If NULL, which is the default condition, no sorting is performed. Sorting only occurs during record insertion and when changing the value of this field. The third parameter of the comparison function, **pStorage**, must be NULL. See CM_SORTRECORD for a further description of the comparison function.

---

pFieldInfoLast

pFieldInfoLast (PFIELDINFO)
Pointer or NULL.

Pointer to last column in the left window of the split details view. The default is NULL, causing all columns to be positioned in the left window.

---

pFieldInfoObject

pFieldInfoObject (PFIELDINFO)
Pointer.

Pointer to a column that represents an object in the details view. The data for this FIELDINFO structure must contain icons or bit maps. In-use emphasis is applied to this column of icons or bit maps only. The default is the leftmost column in the unsplit details view, or the leftmost column in the left window of the split details view.

---

pszCnrTitle

pszCnrTitle (PSZ)
Title text or NULL.

Text for the container title. The default is NULL.

---

flWindowAttr

flWindowAttr (ULONG)
Window attributes.

Consists of container window attributes.

- Specify one of the following container views, which determine the presentation format of items in a container:

  CV_ICON
  In the icon view, the container items are represented as icon/text or bit-map/text pairs, with text beneath the icons or bit maps. This is the default view. This view can be combined with the CV_MINI style bit by using an OR operator (|).

  CV_NAME
  In the name view, the container items are represented as icon/text or bit-map/text pairs, with text to the right of the icons or bit maps. This view can be combined with the CV_MINI and CV_FLOW style bits by using OR operators (|).

  CV_TEXT
  In the text view, the container items are displayed as a list of text strings. This view can be combined with the CV_FLOW style bit by using an OR operator (|).

  CV_TREE
  In the tree view, the container items are represented in a hierarchical manner. The tree view has three forms, which are defined in the following list. If you specify CV_TREE by itself, the tree icon view is used.
  - Tree icon view
    The tree icon view is specified by using a logical OR operator to combine the tree view with the icon view (CV_TREE | CV_ICON). Container items in this view are represented as

icon/text pairs or bit-map/text pairs, with text to the right of the icons or bit maps. Also, a collapsed or expanded icon or bit map is displayed to the left of parent items. If this icon or bit map is a *collapsed* icon or bit map, selecting it will cause the parent item to be expanded so that its child items are displayed below it. If this icon or bit map is an *expanded* icon or bit map, selecting it will cause the parent's child items to be removed from the display. The default collapsed and expanded bit maps provided by the container use a plus sign (+) and a minus sign (-), respectively, to indicate that items can be added to or subtracted from the display.

- Tree name view
  The tree name view is specified by using a logical OR operator to combine the tree view with the name view (CV_TREE | CV_NAME). Container items in this view are displayed as either icon/text pairs or bit-map/text pairs, with text to the right of the icons or bit maps. However, the indicator that represents whether an item can be collapsed or expanded, such as a plus or minus sign, is included in the icon or bit map that represents that item, not in a separate icon or bit map as in the tree icon and tree text views. The container control does not provide default collapsed and expanded bit maps for the tree name view.

- Tree text view
  The tree text view is specified by using a logical OR operator to combine the tree view with the text view (CV_TREE | CV_TEXT). Container items in this view are displayed as a list of text strings. As in the tree icon view, a collapsed or expanded icon or bit map is displayed to the left of parent items.

CV_DETAIL
In the details view, the container items are presented in columns. Each column can contain icons or bit maps, text, numbers, dates, or times.

- Specify one or both of the following view styles by using an OR operator (|) to combine them with the specified view. These view styles are optional.

  CV_MINI
  Produces a mini-icon whose size is based on the Presentation Manager (PM) SV_CYMENU system value to produce a device-dependent mini-icon.
  The CV_MINI view style bit is ignored when:
  - The text view (CV_TEXT), tree view (CV_TREE), or details view (CV_DETAIL) are displayed

  - The CCS_MINIRECORDCORE style bit is specified.

  If this style bit is not specified and the icon view (CV_ICON) or name view (CV_NAME) is used, the default, regular-sized icon is used. The size of regular-sized icons is based on the value in the **slBitmapOrIcon** field of the CNRINFO data structure. If this field is equal to 0, the PM SV_CXICON and SV_CYICON system values for width and height, respectively, are used. Icon sizes are consistent with PM-defined icon sizes for all devices.
  CV_FLOW
  Dynamically arranges container items in columns in the name and text views. These are called flowed name and flowed text views. If this style bit is set for the name view (CV_NAME) or text view (CV_TEXT), the container items are placed in a single column until the bottom of the client area is reached. The next container item is placed in the adjacent column to the right of the filled column. This process is repeated until all of the container items are positioned in the container. The width of each column is determined by the longest text string in that column. The size of the window determines the depth of the client area.
  If this style bit is not specified, the default condition for the name and text views is to vertically fill the container in a single column without flowing the container items. If this style bit is set for the icon view (CV_ICON) or details view (CV_DETAIL), it is ignored.

- Specify either of the following to indicate whether the container will display icons or bit maps:

  CA_DRAWICON
  Icons are used for the icon, name, tree, or details views. This is the default. This container attribute should be used with the **hptrIcon** and **hptrMiniIcon** fields of the RECORDCORE data structure.
  CA_DRAWBITMAP
  Bit maps are used for the icon, name, tree, or details views. This container attribute can be used with the **hbmBitmap** and **hbmMiniBitmap** fields of the RECORDCORE data structure.
  Notes
  1. If both the CA_DRAWICON and CA_DRAWBITMAP attributes are specified, the CA_DRAWICON attribute is used.

  2. If the CCS_MINIRECORDCORE style bit is specified when a container is created, the **hptrIcon** field of the MINIRECORDCORE data structure is used.

- Specify one of the following attributes to provide target emphasis for the name, text, and details views. If neither ordered nor mixed target emphasis is specified, the emphasis is drawn around the record.

CA_ORDEREDTARGETEMPH
: Shows where a container record can be dropped during direct manipulation by drawing a line beneath the record. Ordered target emphasis does not apply to the icon and tree views.

CA_MIXEDTARGETEMPH
: Shows where a container record can be dropped during direct manipulation either by drawing a line between two items or by drawing lines around the container record. Mixed target emphasis does not apply to the icon and tree views.

- Specify the following attribute to draw lines that show the relationship between items in the tree view.

  CA_TREELINE
  : Shows the relationship between all items in the tree view.

- Specify the following to draw container records, paint the background of the container, or both:

  CA_OWNERDRAW
  : **Ownerdraw** for the container, which allows the application to draw container records.

  CA_OWNERPAINTBACKGROUND
  : Allows the application to subclass the container and paint the background. If specified, and the container is subclassed, the application receives the CM_PAINTBACKGROUND message in the subclass procedure. Otherwise, the container paints the background using the color specified by SYSCLR_WINDOW, which can be changed by using the PP_BACKGROUNDCOLOR or PP_BACKGROUNDCOLORINDEX presentation parameter in the WM_PRESPARAMCHANGED (in Container Controls) message.

- Specify the following if the container is to have a title:

  CA_CONTAINERTITLE
  : Allows you to include a container title. The default is no container title.

- Specify one or both of the following container title attributes. These are valid only if the CA_CONTAINERTITLE attribute is specified.

  CA_TITLEREADONLY
  : Prevents the container title from being edited directly. The default is to allow the container title to be edited.

  CA_TITLESEPARATOR
  : Puts a separator line between the container title and the records beneath it. The default is no separator line.

- Specify one of the following to position the container title. These are valid only if the CA_CONTAINERTITLE attribute is specified.

  CA_TITLECENTER
  : Centers the container title. This is the default.

  CA_TITLELEFT
  : Left-justifies the container title.

  CA_TITLERIGHT
  : Right-justifies the container title.

- Specify the following to display column headings in the details view:

  CA_DETAILSVIEWTITLES
  : Allows you to include column headings in the details view. The default is no column headings.

---

ptlOrigin

ptlOrigin (POINTL)
: Workspace origin.

  Lower-left origin of the workspace in virtual coordinates, used in the icon view. The default origin is **(0,0)**.

---

cDelta

cDelta (ULONG)
Threshold

An application-defined threshold, or number of records, from either end of the list of available records. Used when a container needs to handle large amounts of data. The default is 0. Refer to the *OS/2 Programming Guide* for more information about specifying deltas.

---

cRecords

cRecords (ULONG)
Number of records.

The number of records in the container. Initially this field is 0.

---

slBitmapOrIcon

slBitmapOrIcon (SIZEL)
Icon/bit-map size.

The size (in pels) of icons or bit maps. The default is the system size.

---

slTreeBitmapOrIcon

slTreeBitmapOrIcon (SIZEL)
Icon/bit-map size.

The size (in pels) of the expanded and collapsed icons or bit maps used in the tree icon and tree text views.

---

hbmExpanded

hbmExpanded (HBITMAP)
Bit-map handle.

The handle of the bit map to be used to represent an expanded parent item in the tree icon and tree text views. If neither an icon handle (see **hptrExpanded**) nor a bit-map handle is specified, a default bit map with a minus sign (-) is provided.

---

hbmCollapsed

hbmCollapsed (HBITMAP)
Bit-map handle.

The handle of the bit map to be used to represent a collapsed parent item in the tree icon and tree text views. If neither an icon handle (see **hptrCollapsed**) nor a bit-map handle is specified, a default bit map with a plus sign (+) is provided.

---

hptrExpanded

hptrExpanded (HPOINTER)
Icon handle.

The handle of the icon to be used to represent an expanded parent item in the tree icon and tree text views. If neither an icon handle nor a bit-map handle (see **hbmExpanded**) is specified, a default bit map with a minus sign (-) is provided.

---

hptrCollapsed

hptrCollapsed (HPOINTER)
Icon handle.

The handle of the icon to be used to represent a collapsed parent item in the tree icon and tree text views. If neither an icon handle nor a bit-map handle (see **hbmCollapsed**) is specified, a default bit map with a plus sign (+) is provided.

---

cyLineSpacing

cyLineSpacing (LONG)
Vertical space.

The amount of vertical space (in pels) between the records. If you specify a value that is less than 0, a default value is used.

---

cxTreeIndent

cxTreeIndent (LONG)
Horizontal space.

The amount of horizontal space (in pels) between levels in the tree view. If you specify a value that is less than 0, a default value is

used.

---

cxTreeLine

cxTreeLine (LONG)
Line width.

The width of the lines (in pels) that show the relationship between tree items. If you specify a value that is less than 0, a default value is used. Also, if the CA_TREELINE container attribute of the **flWindowAttr** field is not specified, these lines are not drawn.

---

cFields

cFields (ULONG)
Number of columns.

The number of FIELDINFO structures in the container. Initially this field is 0.

---

xVertSplitbar

xVertSplitbar (LONG)
Split bar position.

The initial position of the split bar relative to the container, used in the details view. If this value is less than 0, the split bar is not used. The default value is negative one (-1).

LAYOUT_EDIT_SIZE

LAYOUT_EDIT_SIZE Defines number of surrounding code elements that need to be considered when performing edit shaping.

```
typedef struct  _LAYOUT_EDIT_SIZE {
ULONG    front;      /* Previous characters */
ULONG    back;       /* Succeeding characters */
 } LAYOUT_EDIT_SIZE;
```

---

front

front (ULONG)

Previous characters.

Number of code elements in front of the substring.

---

back

back (ULONG)
Succeeding characters.

Number of code elements behind the substring.

LAYOUT_OBJECT

LAYOUT_OBJECT Bit-map handle.

```
typedef PVOID  LAYOUT_OBJECT;
```

LAYOUT_TEXT_DESCRIPTOR

LAYOUT_TEXT_DESCRIPTOR Identifies the attributes of the source and target text.

```
typedef struct  _LAYOUT_TEXT_DESCRIPTOR {
ULONG    in;      /* Input buffer description */
ULONG    out;     /* Output description */
 } LAYOUT_TEXT_DESCRIPTOR;
```

---

front

in (ULONG)
Input buffer description.

Attributes that describe the input buffer.

---

back

out (ULONG)
Output description.

Attributes that describe the required ouput buffer.

LAYOUT_VALUES

LAYOUT_VALUES Layout object default values.

These values are automatically associated with the layout object at creation time.

```
typedef struct  _LAYOUT_VALUES {
ULONG    name;        /* Name of Layout Value Item */
PVOID    value;       /* Data of Layout Value Item */
 } LAYOUT_VALUES;
```

---

name

name (ULONG)
Name of Layout Value Item.

The following are possible names:

For a more detailed explaination please refer to the section on Layout Values .

Value Name	Value Type	Set/Get
ActiveBidirection	BOOL	G
ActiveShapeEditing	BOOL	G
ShapeContextSize	LAYOUT_EDIT_SIZE	G
Cellsize	ULONG	SG
InputMode	BOOL	SG
CallerAllocMem	BOOL	SG
QueryValueSize	ULONG	SG
InOutTextDescrMask	ULONG	SG
InOnlyTextDescr	ULONG	SG
OutOnlyTextDescr	ULONG	SG
Orientation	LAYOUT_TEXT_DESCRIPTOR	SG
TypeOfText	LAYOUT_TEXT_DESCRIPTOR	SG
Swapping	LAYOUT_TEXT_DESCRIPTOR	SG
Numerals	LAYOUT_TEXT_DESCRIPTOR	SG
TextShaping	LAYOUT_TEXT_DESCRIPTOR	SG
WordBreak	LAYOUT_TEXT_DESCRIPTOR	SG

All of the above can be ORed together except the following.

```
ActiveShapeEditing
ActiveBidirection
ShapeContextSize

CellSize
InputMode
InOnlyTextDescr
OutOnlyTextDescr
InOutTextDescrMask
CallerAllocMem
```

---

value

value (PVOID)
Data of Layout Value Item.

Each value element of a LAYOUT_VALUES record must contain a pointer to the type of the layout value that is being set or get. That is, if the layout value is for type T, the arguement must be of type T*.

ActiveBidirection

```
#define ActiveBidirection       (0x0002<<16)
```

ActiveShapeEditing

```
#define ActiveShapeEditing      (0x0001<<16)
```

ShapeContextSize

```
#define ShapeContextSize        (0x0005<<16)
```

CellSize

```
#define CellSize                (0x0006<<16)
```

InputMode

```
#define InputMode               (0x0007<<16)
```

CallerAllocMem

```
#define CallerAllocMem          (0x000b<<16)
```

QueryValueSize

```
#define QueryValueSize          (0x8000<<16)
```

InOutTextDescrMask

```
#define InOutTextDescrMask      (0x000a<<16)
```

InOnlyTextDescr

```
#define InOnlyTextDescr         (0x0008<<16)
```

OutOnlyTextDescr

```
#define OutOnlyTextDescr        (0x0009<<16)
```

Orientation

```
#define Orientation             0x00000001
```

TypeOfText

```
#define TypeOfText              0x00000002
```

Swapping

```
#define Swapping                0x00000004
```

Numerals

```
#define Numerals                0x00000008
```

TextShaping

```
#define TextShaping             0x00000010
```

Word_Break

```
#define Word_Break              0x00000080
```

QueryValueSize

```
#define QueryValueSize          (0x8000<<16)
```

PLAYOUT_OBJECT

PLAYOUT_OBJECT Bit-map handle.

```
typedef PVOID  FAR *  PLAYOUT_OBJECT;
```

PLAYOUT_VALUES

PLAYOUT_VALUES Bit-map handle.

```
typedef LAYOUT_VALUES FAR *  PLAYOUT_VALUES;
```

CTIME

CTIME Structure that contains time information for a data element in the details view of a container control.

```
typedef struct _CTIME {
UCHAR    hours;       /* Hour  */
UCHAR    minutes;     /* Minute  */
UCHAR    seconds;     /* Second  */
UCHAR    ucReserved;  /* Reserved  */
 } CTIME;
```

---

hours

hours (UCHAR)
Hour.

---

minutes

minutes (UCHAR)
Minute.

---

seconds

seconds (UCHAR)
Second.

---

ucReserved

ucReserved (UCHAR)
Reserved.

FIELDINFO

FIELDINFO Structure that contains information about column data in the details view of the container control. The details view displays each FIELDINFO structure as a column of data that contains specific information about each container record. For example, one FIELDINFO structure, or column, might contain icons or bit maps that represent each container record. Another FIELDINFO structure might contain the date or time that each container record was created.

```
typedef struct _FIELDINFO {
ULONG         cb;              /* Structure size  */
ULONG         flData;          /* Data attributes  */
ULONG         flTitle;         /* Attributes of column headings  */
PVOID         pTitleData;      /* Column heading data  */
ULONG         offStruct;       /* Structure offset  */
PVOID         pUserData;       /* Pointer  */
PFIELDINFO    pNextFieldInfo;  /* Pointer  */
ULONG         cxWidth;         /* Column width  */
 } FIELDINFO;
```

---

cb

cb (ULONG)
Structure size.

The size (in bytes) of the FIELDINFO structure.

---

flData

flData (ULONG)
Data attributes.

Attributes of the data in a field.

- Specify one of the following for each column to choose the type of data that is displayed in each column:

  CFA_BITMAPORICON
  The column contains bit-map or icon data.
  CFA_STRING
  Character or text data is displayed in this column.
  CFA_ULONG
  Unsigned number data is displayed in this column. National Language Support (NLS) is enabled for number format.
  CFA_DATE
  The data in the column is displayed in date format. National Language Support (NLS) is enabled for date format. Use the data structure described in CDATE.
  CFA_TIME
  The data in the column is displayed in time format. National Language Support (NLS) is enabled for time format. Use the data structure described in CTIME.

- Specify any or all of the following column attributes:

  CFA_HORZSEPARATOR
  A horizontal separator is provided beneath column headings.
  CFA_SEPARATOR
  A vertical separator is drawn after this column.
  CFA_OWNER
  Ownerdraw is enabled for this container column.
  CFA_INVISIBLE
  Invisible container column. The default is visible.
  CFA_FIREADONLY
  Prevents text in a FIELDINFO data structure (text in a column) from being edited directly. This attribute applies only to columns for which the CFA_STRING attribute has been specified.

- Specify one of the following for each column to vertically position data in that column:

  CFA_TOP
  Top-justifies field data.
  CFA_BOTTOM
  Bottom-justifies field data.
  CFA_VCENTER
  Vertically centers field data. This is the default.

- Specify one of the following for each column to horizontally position data in that column. These attributes can be combined with the attributes used for vertical positioning of column data by using an OR operator (|).

  CFA_CENTER
  Horizontally centers field data.
  CFA_LEFT
  Left-justifies field data. This is the default.
  CFA_RIGHT
  Right-justifies field data.

---

flTitle

flTitle (ULONG)
Attributes of column headings.

- Specify the following if icon or bit-map data is to be displayed in the column heading:

  CFA_BITMAPORICON
  The column heading contains icon or bit-map data. If CFA_BITMAPORICON is not specified, any data that is assigned to the column heading is assumed to be text or character data.

- Specify the following to prevent direct editing of a column heading:

  CFA_FITITLEREADONLY
  Prevents a column heading from being edited directly.

- Specify one of the following for each column heading to vertically position data in that column heading:

  CFA_TOP
  Top-justifies column headings.
  CFA_BOTTOM
  Bottom-justifies column headings.
  CFA_VCENTER
  Vertically centers column headings. This is the default.

- Specify one of the following for each column heading to horizontally position data in that column heading. These attributes can be combined with the attributes used for vertical positioning of column heading data by using an OR operator (|).

  CFA_CENTER
  Horizontally centers column headings.
  CFA_LEFT
  Left-justifies column headings. This is the default.
  CFA_RIGHT
  Right-justifies column headings.

---

pTitleData

pTitleData (PVOID)
Column heading data.

Column heading data, which can be a text string, or an icon or bit map. The default is a text string. If the **flTitle** field is set to the CFA_BITMAPORICON attribute, this must be an icon or bit map.

---

offStruct

offStruct (ULONG)
Structure offset.

Offset from the beginning of a RECORDCORE structure to the data that is displayed in this column.

Note: If the CCS_MINIRECORDCORE style bit is specified when a container is created, then MINIRECORDCORE should be used instead of RECORDCORE and PMINIRECORDCORE should be used instead of PRECORDCORE in all applicable data structures and messages.

---

pUserData

pUserData (PVOID)
Pointer.

Pointer to user data.

---

pNextFieldInfo

pNextFieldInfo (PFIELDINFO)
Pointer.

Pointer to the next linked FIELDINFO data structure.

---

cxWidth

cxWidth (ULONG)
Column width.

Used to specify the width of a column. The default is an automatically sized column that is always the width of its widest element. If this field is set and the data is too wide, the data is truncated.

---

ulFlags

ulFlags (ULONG)
FNTF_* flags.
FNTF_NOVIEWPRINTERFONTS
This flag is initialized only when both **hpsScreen** and **hpsPrinter** are not NULLHANDLE. On input, this parameter determines whether the printer fonts are to be included in the font list box. The user controls this with a check box.
FNTF_NOVIEWSCREENFONTS
This flag is initialized only when both **hpsScreen** and **hpsPrinter** are not NULLHANDLE. On input, this parameter determines whether the screen fonts should be included in the font list box. The user controls this with a check box.
FNTF_PRINTERFONTSELECTED
This determines if a printer-specific font is selected by the user. The application should make an approximation of this printer font when outputting to the screen. This is an output-only flag and is ignored on input.
FNTF_SCREENFONTSELECTED
This determines if a screen-specific font is selected by the user. The application should make an approximation of this screen font when outputting to the screen. This is an output-only flag and is ignored on input.

HBITMAP

HBITMAP Bit-map handle.

```
typedef LHANDLE HBITMAP;
```

HWND

HWND Window handle.

```
typedef LHANDLE HWND;
```

HFILE

HFILE Resource handle.

```
typedef LHANDLE HFILE;
```

HPOINTER

HPOINTER Pointer handle.

```
typedef LHANDLE HPOINTER;
```

LONG

LONG Signed integer in the range -2 147 483 648 through 2 147 483 647.

Note: Where this data type represents a graphic coordinate in world or model space, its value is restricted to -134 217 728 through 134 217 727.

A graphic coordinate in device or screen coordinates is restricted to -32 768 through 32 767.

The value of a graphic coordinate may be further restricted by any transforms currently in force, including the positioning of the origin of the window on the screen. In particular, coordinates in world or model space must not generate coordinate values after transformation (that is, in device or screen space) outside the range -32 768 through 32 767.

```
#define LONG long
```

MINIRECORDCORE

MINIRECORDCORE Structure that contains information for smaller records than those defined by the RECORDCORE data structure. This data structure is used if the CCS_MINIRECORDCORE style bit is specified when a container is created.

```
typedef struct _MINIRECORDCORE {
ULONG              cb;               /* Structure size  */
ULONG              flRecordAttr;     /* Attributes of container records  */
POINTL             ptlIcon;          /* Record position  */
PMINIRECORDCORE    preccNextRecord;  /* Pointer  */
PSZ                pszIcon;          /* Record text  */
HPOINTER           hptrIcon;         /* Record icon  */
 } MINIRECORDCORE;
```

---

cb

cb (ULONG)
Structure size.

The size (in bytes) of the MINIRECORDCORE structure.

---

flRecordAttr

flRecordAttr (ULONG)
Attributes of container records.

Contains any or all of the following:
CRA_COLLAPSED
Specifies that a record is collapsed.
CRA_CURSORED
Specifies that a record will be drawn with a selection cursor.
CRA_DROPONABLE
Specifies that a record can be a target for direct manipulation.
CRA_EXPANDED
Specifies that a record is expanded.
CRA_FILTERED
Specifies that a record is filtered, and therefore hidden from view.
CRA_INUSE
Specifies that a record will be drawn with in-use emphasis.
CRA_RECORDREADONLY
Prevents a record from being edited directly.
CRA_SELECTED
Specifies that a record will be drawn with selected-state emphasis.
CRA_TARGET
Specifies that a record will be drawn with target emphasis.

---

ptlIcon

ptlIcon (POINTL)
Record position.

Position of a container record in the icon view.

---

preccNextRecord

preccNextRecord (PMINIRECORDCORE)
Pointer.

Pointer to the next linked record.

---

pszIcon

pszIcon (PSZ)
Record text.

Text for the container record.

---

hptrIcon

hptrIcon (HPOINTER)
Record icon.

Icon that is displayed for the container record.

MLE_SEARCHDATA

MLE_SEARCHDATA Search structure for multiline entry field.

```
typedef struct _SEARCH {
USHORT    cb;          /* Size of structure  */
PCHAR     pchFind;     /* Logical string to search for  */
PCHAR     pchReplace;  /* Logical string to replace with  */
SHORT     cchFind;     /* Length of pchFind string  */
SHORT     cchReplace;  /* Length of pchReplace string  */
IPT       iptStart;    /* Visual point at which to start search, or point where string was found  */
IPT       iptStop;     /* Visual point at which to stop search  */
USHORT    cchFound;    /* Length of string found at iptStart  */
 } MLE_SEARCHDATA;
```

---

cb

cb (USHORT)
Size of MLE_SEARCHDATA structure.

---

pchFind

pchFind (PCHAR)
Logical string to search for.

---

pchReplace

pchReplace (PCHAR)
Logical string to replace with.

---

cchFind

cchFind (SHORT)
Length of **pchFind** string.

---

cchReplace

cchReplace (SHORT)
Length of **pchReplace** string.

---

iptStart

iptStart (IPT)
Visual point at which to start search, or point where string was found.
non-negative

Visual point at which to start search.

negative

Start search from current visual cursor location.

---

iptStop

iptStop (IPT)

Visual point at which to stop search.

non-negative

Visual point at which to stop search.

negative

Stop search at end of text.

---

cchFound

cchFound (USHORT)

Length of string found at **iptStart**.

PBD_ATTR_MASK

PBD_ATTR_MASK Pointer to a BD_ATTR_MASK data structure.

```
typedef BD_ATTR_MASK *PBD_ATTR_MASK;
```

---

pBD_Attr_Mask

pBD_Attr_Mask (PBD_ATTR_MASK) pointer.

Pointer to the BD_ATTR_MASK data structure.

PCHAR

PCHAR Pointer to CHAR.

```
typedef CHAR *PCHAR;
```

PFIELDINFO

PFIELDINFO Pointer to a FIELDINFO data structure.

```
typedef FIELDINFO *PFIELDINFO;
```

POINTL

POINTL Point structure (long integer).

```
typedef struct _POINTL {
LONG    x;  /* x-coordinate  */
LONG    y;  /* y-coordinate  */
 } POINTL;
```

---

x

x (LONG)
x-coordinate.

---

y

y (LONG)
y-coordinate.

PSZ

PSZ Pointer to a null-terminated string.

```
typedef char *PSZ;
```

PTREEITEMDESC

PTREEITEMDESC Pointer to a TREEITEMDESC data structure.

```
typedef TREEITEMDESC *PTREEITEMDESC;
```

---

pFieldInfo

pFieldInfo (PFIELDINFO)
Pointer.

Pointer to the FIELDINFO structure for the container column that is being drawn in the details view. For all other views, this field is NULL.

PMINIRECORDCORE

PMINIRECORDCORE Pointer to a MINIRECORDCORE data structure.

```
typedef MINIRECORDCORE *PMINIRECORDCORE;
```

PRECORDCORE

PRECORDCORE Pointer to a RECORDCORE data structure.

```
typedef RECORDCORE *PRECORDCORE;
```

---

pRecordcore

pRecordcore (PRECORDCORE) pointer

Pointer to the RECORDCORE data structure.

PUCHAR

PUCHAR Pointer to UCHAR.

```
typedef UCHAR *PUCHAR;
```

---

pulAttr

pulAttr Pointer to a BIDIATTR data structure.

PULONG

PULONG Pointer to ULONG.

```
typedef ULONG *PULONG;
```

PVOID

PVOID Pointer to a data type of undefined format.

```
typedef VOID *PVOID;
```

RECORDCORE

RECORDCORE Structure that contains information for records in a container control. This data structure is used if the CCS_MINIRECORDCORE style bit is not specified when a container is created.

```
typedef struct _RECORDCORE {
ULONG           cb;              /* Structure size  */
ULONG           flRecordAttr;    /* Record attributes  */
POINTL          ptlIcon;         /* Record position  */
PRECORDCORE     preccNextRecord; /* Pointer  */
PSZ             pszIcon;         /* Text  */
HPOINTER        hptrIcon;        /* Icon  */
HPOINTER        hptrMiniIcon;    /* Mini-icon  */
HBITMAP         hbmBitmap;       /* Bit map  */
HBITMAP         hbmMiniBitmap;   /* Mini-bit map  */
PTREEITEMDESC   pTreeItemDesc;   /* Pointer  */
PSZ             pszText;         /* Text view text  */
PSZ             pszName;         /* Name view text  */
PSZ             pszTree;         /* Tree view text  */
 } RECORDCORE;
```

---

cb

cb (ULONG)
Structure size.

The size (in bytes) of the RECORDCORE structure.

---

flRecordAttr

flRecordAttr (ULONG)
Record attributes.

Attributes of container records. Contains any or all of the following:
CRA_COLLAPSED
Specifies that a record is collapsed.
CRA_CURSORED
Specifies that a record will be drawn with a selection cursor.
CRA_DROPONABLE
Specifies that a record can be a target for direct manipulation.
CRA_EXPANDED
Specifies that a record is expanded.
CRA_FILTERED
Specifies that a record is filtered, and therefore hidden from view.
CRA_INUSE
Specifies that a record will be drawn with in-use emphasis.
CRA_RECORDREADONLY
Prevents a record from being edited directly.
CRA_SELECTED
Specifies that a record will be drawn with selected-state emphasis.
CRA_TARGET
Specifies that a record will be drawn with target emphasis.

---

ptlIcon

ptlIcon (POINTL)
Record position.

Position of a container record in the icon view.

---

preccNextRecord

preccNextRecord (PRECORDCORE)
Pointer.

Pointer to the next linked record.

---

pszIcon

pszIcon (PSZ)
Text.

Text for the icon view (CV_ICON).

---

hptrIcon

hptrIcon (HPOINTER)
Icon.

Icon that is displayed when the CV_MINI style bit is not specified. This field is used when the CA_DRAWICON container attribute of the CNRINFO data structure is set.

---

hptrMiniIcon

hptrMiniIcon (HPOINTER)
Mini-icon.

Icon that is displayed when the CV_MINI style bit is specified. This field is used when the CA_DRAWICON container attribute of the CNRINFO data structure is set.

---

hbmBitmap

hbmBitmap (HBITMAP)
Bit map.

Bit map that is displayed when the CV_MINI style bit is not specified. This field is used when the CA_DRAWBITMAP container attribute of the CNRINFO data structure is set.

---

hbmMiniBitmap

hbmMiniBitmap (HBITMAP)
Mini-bit map.

Bit map that is displayed when the CV_MINI style bit is specified. This field is used when the CA_DRAWBITMAP container attribute of the CNRINFO data structure is set.

---

pTreeItemDesc

pTreeItemDesc (PTREEITEMDESC)
Pointer.

Pointer to a TREEITEMDESC structure, which contains the icons and bit maps used to represent the state of an expanded or collapsed parent item in the tree name view.

---

pszText

pszText (PSZ)
Text view text.

Text for the text view (CV_TEXT).

---

pszName

pszName (PSZ)
Name view text.

Text for the name view (CV_NAME).

---

pszTree

pszTree (PSZ)
Tree view text.

Text for the tree view (CV_TREE).

SIZEL

SIZEL Size structure.

```
typedef struct _SIZEL {
LONG    cx;  /* Width  */
LONG    cy;  /* Height  */
 } SIZEL;
```

---

cx

cx (LONG)
Width.

---

cy

cy (LONG)
Height.

TREEITEMDESC

TREEITEMDESC Structure that contains icons and bit maps used to represent the state of an expanded or collapsed parent item in the tree name view of a container control.

```
typedef struct _TREEITEMDESC {
HBITMAP     hbmExpanded;    /* Expanded bit-map handle  */
HBITMAP     hbmCollapsed;   /* Collapsed bit-map handle  */
HPOINTER    hptrExpanded;   /* Expanded icon handle  */
HPOINTER    hptrCollapsed;  /* Collapsed icon handle  */
 } TREEITEMDESC;
```

HAB

HAB Anchor-block handle.

```
typedef LHANDLE HAB;
```

---

hbmExpanded

hbmExpanded (HBITMAP)
Expanded bit-map handle.

The handle of the bit map to be used to represent an expanded parent item in the tree name view.

---

hbmCollapsed

hbmCollapsed (HBITMAP)
Collapsed bit-map handle.

The handle of the bit map to be used to represent a collapsed parent item in the tree name view.

---

hptrExpanded

hptrExpanded (HPOINTER)
Expanded icon handle.

The handle of the icon to be used to represent an expanded parent item in the tree name view.

---

hptrCollapsed

hptrCollapsed (HPOINTER)
Collapsed icon handle.

The handle of the icon to be used to represent a collapsed parent item in the tree name view.

HWND

HWND Window handle.

```
typedef LHANDLE HWND;
```

UCHAR

UCHAR Unsigned integer in the range 0 through 255.

```
typedef unsigned char UCHAR;
```

---

ulData

ulData (ULONG)

User-defined value.

ULONG

ULONG Unsigned integer in the range 0 through 4 294 967 295.

```
typedef unsigned long ULONG;
```

---

ulNotify

The following is a list of possible values for these events:

```
LVI_WND_ATTR       - Window Bidi attributes changed.
LVI_WND_STAT       - Window Bidi status changed.
LVI_FOCUS_CHANGED  - The focus window has changed.
LVI_KBD_LAYER      - The system keyboard layer has changed.
LVI_ALL            - All information in the viewer must be updated.
LVI_HIDE           - This value is used when the system notifies the
                     viewer to hide itself (such as when another viewer
                     is the active one).
```

---

ulPageId

ulPageId (ULONG)
Page identifier.

Page identifier used for placement of the inserted page.

---

ulPageId

ulPageId (ULONG)
Page id

Page identifier of the page whose tab text is requested.

---

ulPageId

ulPageId (ULONG)
Page id

Page identifier of the page whose Status line text is requested.

USHORT

USHORT Unsigned integer in the range 0 through 65 535.

```
typedef unsigned short USHORT;
```

CM_SHORTRECORD

PLEASE REFER TO THE PM REFERNCE.

WM_PRESPARAMCHANGED

PLEASE REFER TO THE PM REFERNCE.

CM_PAINTBACKGROUND

PLEASE REFER TO THE PM REFERNCE.

WinGetLastError

PLEASE REFER TO THE PM REFERNCE.

ERROR_INVALID_LENGTH

PLEASE REFER TO THE PM REFERNCE.

BIDI_STAT_VALUES

```
Possible value for Bidi Status :

#define BDS_HKFLAG_ENG_LAYER            0x00010000UL
#define BDS_HKFLAG_NAT_LAYER            0x00020000UL
#define BDS_HKFLAG_PUSH                 0x00040000UL
#define BDS_HKFLAG_END_PUSH             0x00080000UL
#define BDS_HKFLAG_AUTO_PUSH            0x00100000UL
#define BDS_HKFLAG_FIELD_REV            0x00200000UL
#define BDS_HKFLAG_SCREEN_REV           0x00400000UL
#define BDS_HKFLAG_BIDI_POPUP           0x02000000UL
#define BDS_HKFLAG_AUTOMATIC            0x04000000UL
#define BDS_HKFLAG_INITIAL              0x08000000UL
#define BDS_HKFLAG_MIDDLE               0x10000000UL
#define BDS_HKFLAG_FINAL                0x20000000UL
#define BDS_HKFLAG_ISOLATED             0x40000000UL

#define BDS_FAUTOPUSH_RTL_ON            0x00000001UL
#define BDS_FAUTOPUSH_LTR_ON            0x00000002UL
#define BDS_FPUSH_ON                    0x00000004UL
#define BDS_FKBD_LAYER_IS_NL            0x00000010UL
```

LI_BD_CLIP_ATTR

When ulEffect is set to LI_BD_CLIP_ATTR the ulMask value is ignored and ulData may be set to the following attributes that describe the text in the clipboard:

- **uldata**

```
#define BDA_LEVEL                   0x30000000UL
#define BDA_INIT                    0x80000000UL

#define BDA_DATATYPE_VISUAL         0x00000000UL
#define BDA_DATATYPE_IMPLICIT       0x01000000UL

#define BDA_TEXT_ORIENT_LTR         0x00000000UL
#define BDA_TEXT_ORIENT_RTL         0x00010000UL
#define BDA_TEXT_ORIENT_CONTEXT     0x00020000UL

#define BDA_WND_ORIENT_LTR          0x00000000UL
#define BDA_WND_ORIENT_RTL          0x00100000UL

#define BDA_NUMERALS_ARABIC         0x00000000UL
#define BDA_NUMERALS_WESTERN        0x00000000UL
#define BDA_NUMERALS_PASSTHRU       0x00001000UL
#define BDA_NUMERALS_HINDI          0x00002000UL
#define BDA_NUMERALS_CONTEXT        0x00003000UL

#define BDA_WORDBREAK_OFF           0x00000000UL
#define BDA_WORDBREAK_ON            0x00000200UL
#define BDA_SYM_SWAP_OFF            0x00000000UL
#define BDA_SYM_SWAP_ON             0x00000100UL

#define BDA_CSD_ON                  0x00000000UL
#define BDA_CSD_PASSTHRU            0x00000001UL
#define BDA_CSD_BASE                0x00000010UL
#define BDA_CSD_INITIAL             0x00000011UL
#define BDA_CSD_MIDDLE              0x00000012UL
#define BDA_CSD_FINAL               0x00000013UL
#define BDA_CSD_ISOLATED            0x00000014UL
```

LI_BD_CLIP_CONV_ATTR

When ulEffect is set to LI_BD_CLIP_CONV_ATTR the ulMask value is ignored and ulData may be set to the following attributes of the window that queries text data from the clipboard:

- **uldata**

```
#define BDA_LEVEL                  0x30000000UL
#define BDA_INIT                   0x80000000UL

#define BDA_DATATYPE_VISUAL        0x00000000UL
#define BDA_DATATYPE_IMPLICIT      0x01000000UL

#define BDA_TEXT_ORIENT_LTR        0x00000000UL
#define BDA_TEXT_ORIENT_RTL        0x00010000UL
#define BDA_TEXT_ORIENT_CONTEXT    0x00020000UL

#define BDA_WND_ORIENT_LTR         0x00000000UL
#define BDA_WND_ORIENT_RTL         0x00100000UL

#define BDA_NUMERALS_ARABIC        0x00000000UL
#define BDA_NUMERALS_WESTERN       0x00000000UL
#define BDA_NUMERALS_PASSTHRU      0x00001000UL
#define BDA_NUMERALS_HINDI         0x00002000UL
#define BDA_NUMERALS_CONTEXT       0x00003000UL

#define BDA_WORDBREAK_OFF          0x00000000UL
#define BDA_WORDBREAK_ON           0x00000200UL
#define BDA_SYM_SWAP_OFF           0x00000000UL
#define BDA_SYM_SWAP_ON            0x00000100UL

#define BDA_CSD_ON                 0x00000000UL
#define BDA_CSD_PASSTHRU           0x00000001UL
#define BDA_CSD_BASE               0x00000010UL
#define BDA_CSD_INITIAL            0x00000011UL
#define BDA_CSD_MIDDLE             0x00000012UL
#define BDA_CSD_FINAL              0x00000013UL
#define BDA_CSD_ISOLATED           0x00000014UL
```

LI_BD_PROCESS_ATTR

When ulEffect is set to LI_BD_PROCESS_ATTR the ulMask value is ignored and ulData may be set to the following process attributes:

- **uldata**

```
#define BDA_LEVEL                  0x30000000UL
#define BDA_INIT                   0x80000000UL

#define BDA_DATATYPE_VISUAL        0x00000000UL
#define BDA_DATATYPE_IMPLICIT      0x01000000UL

#define BDA_TEXT_ORIENT_LTR        0x00000000UL
#define BDA_TEXT_ORIENT_RTL        0x00010000UL
#define BDA_TEXT_ORIENT_CONTEXT    0x00020000UL

#define BDA_WND_ORIENT_LTR         0x00000000UL
#define BDA_WND_ORIENT_RTL         0x00100000UL

#define BDA_NUMERALS_ARABIC        0x00000000UL
#define BDA_NUMERALS_WESTERN       0x00000000UL
#define BDA_NUMERALS_PASSTHRU      0x00001000UL
#define BDA_NUMERALS_HINDI         0x00002000UL
#define BDA_NUMERALS_CONTEXT       0x00003000UL

#define BDA_WORDBREAK_OFF          0x00000000UL
#define BDA_WORDBREAK_ON           0x00000200UL
#define BDA_SYM_SWAP_OFF           0x00000000UL
#define BDA_SYM_SWAP_ON            0x00000100UL
```

```
#define BDA_CSD_ON                0x00000000UL
#define BDA_CSD_PASSTHRU          0x00000001UL
#define BDA_CSD_BASE              0x00000010UL
#define BDA_CSD_INITIAL           0x00000011UL
#define BDA_CSD_MIDDLE            0x00000012UL
#define BDA_CSD_FINAL             0x00000013UL
#define BDA_CSD_ISOLATED          0x00000014UL
```

LI_BD_PROCESS_STAT

When ulEffect is set to LI_BD_PROCESS_STAT the ulMask value is ignored and ulData may be set to the following process status:

- **uldata**

```
#define BDS_HKFLAG_ENG_LAYER            0x00010000UL
#define BDS_HKFLAG_NAT_LAYER            0x00020000UL
#define BDS_HKFLAG_PUSH                 0x00040000UL
#define BDS_HKFLAG_END_PUSH             0x00080000UL
#define BDS_HKFLAG_AUTO_PUSH            0x00100000UL
#define BDS_HKFLAG_FIELD_REV            0x00200000UL
#define BDS_HKFLAG_SCREEN_REV           0x00400000UL
#define BDS_HKFLAG_BIDI_POPUP           0x02000000UL
#define BDS_HKFLAG_AUTOMATIC            0x04000000UL
#define BDS_HKFLAG_INITIAL              0x08000000UL
#define BDS_HKFLAG_MIDDLE               0x10000000UL
#define BDS_HKFLAG_FINAL                0x20000000UL
#define BDS_HKFLAG_ISOLATED             0x40000000UL

#define BDS_FAUTOPUSH_RTL_ON            0x00000001UL
#define BDS_FAUTOPUSH_LTR_ON            0x00000002UL
#define BDS_FPUSH_ON                    0x00000004UL
#define BDS_FKBD_LAYER_IS_NL            0x00000010UL
```

LI_BD_WND_ATTR

When ulEffect is set to BD_WND_ATTR the possible values of uldata and ulMask are set to the following:

- **uldata**

```
#define BDA_LEVEL                  0x30000000UL
#define BDA_INIT                   0x80000000UL

#define BDA_DATATYPE_VISUAL        0x00000000UL
#define BDA_DATATYPE_IMPLICIT      0x01000000UL

#define BDA_TEXT_ORIENT_LTR        0x00000000UL
#define BDA_TEXT_ORIENT_RTL        0x00010000UL
#define BDA_TEXT_ORIENT_CONTEXT    0x00020000UL

#define BDA_WND_ORIENT_LTR         0x00000000UL
#define BDA_WND_ORIENT_RTL         0x00100000UL

#define BDA_NUMERALS_ARABIC        0x00000000UL
#define BDA_NUMERALS_WESTERN       0x00000000UL
#define BDA_NUMERALS_PASSTHRU      0x00001000UL
#define BDA_NUMERALS_HINDI         0x00002000UL
#define BDA_NUMERALS_CONTEXT       0x00003000UL

#define BDA_WORDBREAK_OFF          0x00000000UL
```

```
#define BDA_WORDBREAK_ON           0x00000200UL
#define BDA_SYM_SWAP_OFF           0x00000000UL
#define BDA_SYM_SWAP_ON            0x00000100UL

#define BDA_CSD_ON                 0x00000000UL
#define BDA_CSD_PASSTHRU           0x00000001UL
#define BDA_CSD_BASE               0x00000010UL
#define BDA_CSD_INITIAL            0x00000011UL
#define BDA_CSD_MIDDLE             0x00000012UL
#define BDA_CSD_FINAL              0x00000013UL
#define BDA_CSD_ISOLATED           0x00000014UL
```

- **ulmask**

```
#define BDAM_INIT               0x80000000UL
#define BDAM_DATATYPE           0x01000000UL
#define BDAM_TEXT_ORIENTATION   0x00030000UL
#define BDAM_WND_ORIENTATION    0x00100000UL
#define BDAM_NUMERALS           0x00003000UL
#define BDAM_SYM_SWAP           0x00000100UL
#define BDAM_CSD                0x000000FFUL

#define BDAM_ALL               (BDAM_DATATYPE         |
                                BDAM_TEXT_ORIENTATION |
                                BDAM_WND_ORIENTATION  |
                                BDAM_NUMERALS         |
                                BDAM_SYM_SWAP         |
                                BDAM_CSD               )
```

LI_BD_WND_STAT

When ulEffect is set to LI_BD_WND_STAT the possible values of uldata and ulMask are set to the following:

- **uldata**

```
#define BDS_HKFLAG_ENG_LAYER            0x00010000UL
#define BDS_HKFLAG_NAT_LAYER            0x00020000UL
#define BDS_HKFLAG_PUSH                 0x00040000UL
#define BDS_HKFLAG_END_PUSH             0x00080000UL
#define BDS_HKFLAG_AUTO_PUSH            0x00100000UL
#define BDS_HKFLAG_FIELD_REV            0x00200000UL
#define BDS_HKFLAG_SCREEN_REV           0x00400000UL
#define BDS_HKFLAG_BIDI_POPUP           0x02000000UL
#define BDS_HKFLAG_AUTOMATIC            0x04000000UL
#define BDS_HKFLAG_INITIAL              0x08000000UL
#define BDS_HKFLAG_MIDDLE               0x10000000UL
#define BDS_HKFLAG_FINAL                0x20000000UL
#define BDS_HKFLAG_ISOLATED             0x40000000UL

#define BDS_FAUTOPUSH_RTL_ON            0x00000001UL
#define BDS_FAUTOPUSH_LTR_ON            0x00000002UL
#define BDS_FPUSH_ON                    0x00000004UL
#define BDS_FKBD_LAYER_IS_NL            0x00000010UL
```

- **ulMask**

```
#define BDSM_HKFLAGS                    0x7E7F0000UL

#define BDSM_AUTOPUSH_RTL               0x00000001UL
#define BDSM_AUTOPUSH_LTR               0x00000002UL
#define BDSM_PUSH_ON                    0x00000004UL
#define BDSM_KBD_LAYER                  0x00000010UL

#define BDSM_ALL                        0x7E7F0017UL
```

---

ulMask

ulMask (ULONG)

Contains one of the following Bidi attribute mask values:

```
#define BDAM_DATATYPE           0x01000000UL
#define BDAM_TEXT_ORIENTATION   0x00030000UL
#define BDAM_WND_ORIENTATION    0x00100000UL
#define BDAM_NUMERALS           0x00003000UL
#define BDAM_SYM_SWAP           0x00000100UL
#define BDAM_CSD                0x000000FFUL

#define BDAM_ALL               (BDAM_DATATYPE         |
                                BDAM_TEXT_ORIENTATION |
                                BDAM_WND_ORIENTATION  |
                                BDAM_NUMERALS         |
                                BDAM_SYM_SWAP         |
                                BDAM_CSD               )
```

---

ulMask

ulMask (ULONG)

Contains one of the following Bidi status mask values:

```
#define BDSM_HKFLAGS                    0x7E7F0000UL

#define BDSM_AUTOPUSH_RTL               0x00000001UL
#define BDSM_AUTOPUSH_LTR               0x00000002UL
#define BDSM_PUSH_ON                    0x00000004UL
#define BDSM_KBD_LAYER                  0x00000010UL

#define BDSM_ALL                        0x7E7F0017UL
```

BIDIFLAGS

ulFlags (ULONG)

Contains one of the following flag values:

```
#define LIF_NO_SENDMSG        0x00000001UL
#define LIF_CHILD_INHERIT     0x00000002UL
#define LIF_WND_REFRESH       0x00000004UL
```

BIDIFLAGS

ulFlags (ULONG)

Contains one of the following flag values:

OL
: When this is specified the function sends (internally) a WM_QUERYBIDIATTR (or WM_QUERYBIDISTAT) message to the return value procedure. If the return value of the message is other than zero this is the value Returned.

LIF_NO_SENDMSG
: When this value is specified, the WinQueryBidiInfo function, does not send a WM_QUERYBIDIATTR (or WM_QUERYBIDISTAT) message to the window procedure of hwnd. Instead, it returns the Bidi attributes/status as it is stored in the window internal strucrure.

---

ulBD_ATTR

ulBD_ATTR can take any of the values in the BIDIATTR.

---

ulBD_STAT

ulBD_STAT can take any of the values in the BIDI_STAT.

---

ulBD_Mask

ulBD_Mask can take any of the following values When used in the WM_LANGVIEWINFOCHANGED message.

```
#define BDAM_DATATYPE           0x01000000UL
#define BDAM_TEXT_ORIENTATION   0x00030000UL
#define BDAM_WND_ORIENTATION    0x00100000UL
#define BDAM_NUMERALS           0x00003000UL
#define BDAM_SYM_SWAP           0x00000100UL
#define BDAM_CSD                0x000000FFUL

#define BDAM_ALL               (BDAM_DATATYPE         |
                                BDAM_TEXT_ORIENTATION |
                                BDAM_WND_ORIENTATION  |
                                BDAM_NUMERALS         |
                                BDAM_SYM_SWAP         |
                                BDAM_CSD               )
```

---

usColumn

usColumn (USHORT)

Column index

The index of the column over which the direct manipulation action occurred.

---

usRow

usRow (USHORT)

Row index

The index of the row over which the direct manipulation action occurred.

---

ulData

ulData (ULONG)

Contains one of the values inBIDIATTR.

---

ulData

ulData (ULONG)

Contains one of the values inBIDI_STAT.

BOOKPAGEBIDIINFO

BOOKPAGEBIDIINFO

The BOOKPAGEBIDIINFO structure contains the Bidi attributes of the notebook major and minor tabs and of the status line.

```
typedef struct _BOOKPAGEBIDIINFO
{
   BD_ATTR_MASK  bamMajorTab;
   BD_ATTR_MASK  bamMinorTab;
   BD_ATTR_MASK  bamStatusLine;
} BOOKPAGEBIDIINFO;
```

PBOOKPAGEBIDIINFO

PBOOKPAGEBIDIINFO

Pointer to BOOKPAGEBIDIINFO.

```
typedef BOOKPAGEBIDIINFO * PBOOKPAGEBIDIINFO;
```

PVSCDATA

PVSCDATA Pointer to VSCDATA.

```
typedef VSCDATA *PVSCDATA;
```

VSCDATA

VSCDATA Structure that contains information about the value set control.

```
typedef struct _VSCDATA {
ULONG     cbSize;         /* Data length  */
USHORT    usRowCount;     /* Number of rows  */
USHORT    usColumnCount;  /* Number of columns  */
 } VSCDATA;
```

---

cbSize

cbSize (ULONG)
Data length.

Length of the control data in bytes.

---

usRowCount

usRowCount (USHORT)
Number of rows.

The number of rows in the value set control. The minimum number of rows is 1 and the maximum number of rows is 65,535.

---

usColumnCount

usColumnCount (USHORT)
Number of columns.

The number of columns in the value set control. The minimum number of columns is 1 and the maximum number of columns is 65,535.

USHORT

USHORT Unsigned integer in the range 0 through 65 535.

```
typedef unsigned USHORT UUSHORT;
```

IPT

IPT Insertion point for multi-line entry field.

```
typedef LONG IPT;
```

PBUFFER

PBUFFER Pointer to PBYTE.

```
typedef BUFFER *PBUFFER;
```

PFATTRS

PFATTRS Pointer to FATTRS

```
typedef FATTRS *PFATTRS;
```

PIPT

PIPT Pointer to IPT

```
typedef IPT *PIPT;
```

PIX

PIX Pel count for multi-line entry field.

```
typedef LONG PIX;
```

PMLE_SEARCHDATA

PMLE_SEARCHDATA Pointer to an MLE_SEARCHDATA data structure.

```
typedef LONG *PMLE_SEARCHDATA;
```

PPOINTL

PPOINTL Pointer to a POINTL data structure.

```
typedef POINTL *POINTL;
```

Message Processing

Window Messages in the PM Bidirectional environment are processed by window and dialog procedures. Please refer to the explanation of message processing in the PM reference.

Container Control

Processing of the Container control messages in the PM Bidirectional environment is the same as in the Non-Bidirectional PM environment. However, the Container control is sensitive to its Bidirectional attributes and handles formatting and presentation accordingly.

The container supports all text Bidi attributes. This support is done in a hierarchy so that every level (class) of container sub-levels inherits the container attributes by default. These defaults attributes may be overridden by a Bidi-Aware application.

The bidi attributes hierarchy of the container controls works as follows:

> The Container Field bidi attributes can be set by sending a CM_SETFIELDBIDIATTR message to the container. If the field bidi attributes word is not specifially set by sending the message - the field uses its (parent) record bidi attributes.

The Container Record bidi attributes can be set by sending the CM_SETITEMBIDIATTR message. If the record bidi attributes are not specifically set by sending the message - the record uses the container window bidi attributes.

The above inheritance chain defines the effective bidi attribute that is used in processing of a container object.

CM_QUERYFIELDBIDIATTR

Topics - CM_QUERYFIELDBIDIATTR

Select an item:

MAIN - CM_QUERYFIELDBIDIATTR

Cause:

This message queries the bidirectional attributes of a specific Container Field.

Parameters:

```
param1
    PFIELDINFO  pFieldInfo    Pointer.

param2
    PULONG      pBidiAttr     Pointer.

returns
    BOOL        fSuccess      Success indicator.
```

PARAMETERS - CM_QUERYFIELDBIDIATTR

pFieldInfo (PFIELDINFO)
Pointer.

Pointer to the FIELDINFO data structure.

pBidiAttr (PULONG)
Pointer.

Pointer to the values of the Bidi attributes that are returned. Possible values are in the BIDIATTR data structure.

---

pFieldInfo

pFieldInfo (PFIELDINFO)
Pointer.

Pointer to the FIELDINFO data structure.

---

pBidiAttr

pBidiAttr (PULONG)
Pointer.

Pointer to the values of the Bidi attributes that are returned. Possible values are in the BIDIATTR data structure.

RETURN VALUES - CM_QUERYFIELDBIDIINFO

fSuccess(BOOL)
Success indicator.
TRUE
Query of the container bidi information succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Query of the container bidi information succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - CM_QUERYFIELDBIDIATTR

This is the main expl.of cm NOTES

DEFAULT PROCESSING - CM_QUERYFIELDBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

CM_QUERYITEMBIDIATTR

Topics - CM_QUERYITEMBIDIATTR

Select an item:

MAIN - CM_QUERYITEMBIDIATTR

Cause:

This message queries the bidirectional attributes of a specific Container Item (record).

Parameters:

```
param1
    PRECORDCORE  precordcore    Pointer.

param2
    PULONG       pBidiAttr      Pointer.

returns
    BOOL         fsuccess       Success indicator.
```

PARAMETERS - CM_QUERYITEMBIDIATTR

pRecordCore (PRECORDCORE)

Pointer.

Pointer to the RECORDINFO data structure.

pBidiAttr (PULONG)
Pointer.

Pointer to the values of the Bidi attributes that are returned. Possible values are in the BIDIATTR data structure.

---

pRecordCore

pRecordCore (PRECORDCORE)
Pointer.

Pointer to the RECORDINFO data structure.

---

pBidiAttr

pBidiAttr (PULONG)
Pointer.

Pointer to the values of the Bidi attributes that are returned. Possible values are in the BIDIATTR data structure.

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Query of the bidi item information succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

RETURN VALUES - CM_QUERYITEMBIDIINFO

fSuccess(BOOL)
Success indicator.
TRUE
Query of the bidi item information succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - CM_QUERYITEMBIDIATTR

This is the main expl.of cm NOTES

DEFAULT PROCESSING - CM_QUERYITEMBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

CM_SETFIELDBIDIATTR

Topics - CM_SETFIELDBIDIATTR

Select an item:

MAIN - CM_SETFIELDBIDIATTR

Cause:

This message sets the bidirectional attributes of a specific Container Field.

Parameters:

```
param1
    PFIELDINFO        pFieldInfo        Pointer.

param2
    PBD_ATTR_MASK     pBD_Attr_Mask     Pointer.

returns
    BOOL              fsuccess          Success indicator.
```

PARAMETERS - CM_SETFIELDBIDIATTR

pFieldInfo (PFIELDINFO)
Pointer.

Pointer to the FIELDINFO data structure.

pBD_Attr_Mask (PBD_ATTR_MASK)
Pointer.

Pointer to the BD_ATTR_MASK data structure.

---

pFieldInfo

pFieldInfo (PFIELDINFO)
Pointer.

Pointer to the FIELDINFO data structure.

---

pBD_Attr_Mask

pBD_Attr_Mask (PBD_ATTR_MASK)
Pointer.

Pointer to the BD_ATTR_MASK data structure.

RETURN VALUES - CM_SETFIELDBIDIATTR

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the field bidi attribute succeeded.
FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the field bidi attribute succeeded.
FALSE
An error occurred. The WinGetLastError function may return the following error:
- PMERR_INVALID_PARAMETERS

NOTES - CM_SETFIELDBIDIATTR

This is the main expl.of cm NOTES

DEFAULT PROCESSING - CM_SETFIELDBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

CM_SETITEMBIDIATTR

Topics - CM_SETITEMBIDIATTR

Select an item:

MAIN - CM_SETITEMBIDIATTR

Cause:

This message sets the bidirectional attributes of a specific Container Item (record).

Parameters:

```
param1
    PRECORDCORE     precordcore     Pointer.

param2
    PBD_ATTR_MASK   pBD_Attr_Mask   Pointer.

returns
```

```
BOOL            fsuccess        Success indicator.
```

PARAMETERS - CM_SETITEMBIDIATTR

pRecordCore (PRECORDCORE)
Pointer.

Pointer to the RECORDCORE data structure.

pBD_Attr_Mask (PBD_ATTR_MASK)
Pointer.

Pointer to the BD_ATTR_MASK data structure.

---

pRecordCore

pRecordCore (PRECORDCORE)
Pointer.

Pointer to the RECORDCORE data structure.

---

pBD_Attr_Mask

pBD_Attr_Mask (PBD_ATTR_MASK)
Pointer.

Pointer to the BD_ATTR_MASK data structure.

RETURN VALUES - CM_SETITEMBIDIATTR

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the bidi item attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the bidi item attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - CM_SETITEMBIDIATTR

This is the main expl.of cm NOTES

DEFAULT PROCESSING - CM_SETITEMBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

Multi-Line Entry Field Control

Processing of Multi-Line Entry Field control messages in the PM Bidirectional environment is the same as in the non-bidirectional PM environment. However, the Multi-Line Entry Field control is sensitive to its Bidirectional attributes and handles formatting and presentation accordingly.

MLM_CLEAR

Topics - MLM_CLEAR

Select an item:

Main Panel
Parameters
Return Values
Notes
Default Processing

Main - MLM_CLEAR

Cause:

This message clears the currently selected text. When the text type of the MLE is implicit, the characters affected may not necessarily be adjacent in the text buffer : the action is taken on the visually selected text which is continous on the screen.

Parameters:

```
param1
    ULONG param1  Reserved.

param2
    ULONG param2  Reserved.

returns
    ULONG ulClear Number of bytes deleted, counted in CF_TEXT format.
```

Notes - MLM_CLEAR

The multi-line entry field control window procedure responds to this message by clearing the current visual selection and returning the number of bytes cleared.

Default Processing - MLM_CLEAR

The default window procedure takes no action on this message, other than to set *ulClear* to 0.

Parameters - MLM_CLEAR

param1 (ULONG)
Reserved.
0
Reserved value, 0.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_CLEAR

ulClear (ULONG)
Number of bytes deleted, counted in CF_TEXT format.

---

param1

param1 (ULONG)
Reserved.
0
Reserved value, 0.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

ulClear

ulClear (ULONG)
Number of bytes deleted, counted in CF_TEXT format.

MLM_COPY

Topics - MLM_COPY

Select an item:

Main - MLM_COPY

Cause:

This message copies the currently selected text to the clipboard. When the text type of the MLE is implicit, the characters affected may not necessarily be adjacent in the text buffer : the action is taken on the visually selected text which is continous on the screen.

Parameters:

```
param1
    ULONG param1 Reserved.

param2
    ULONG param2 Reserved.

returns
    ULONG ulCopy Number of bytes transferred, counted in CF_TEXT format.
```

Notes - MLM_COPY

The multi-line entry field control window procedure responds to this message by copying the visually selected text to the clipboard. The text is translated to standard clipboard format, which is the same as exporting with MLE_CFTEXT format.

The text is placed on the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64KB).

This may cause an overflow, see MLN_OVERFLOW.

Default Processing - MLM_COPY

The default window procedure takes no action on this message, other than to set *ulCopy* to 0.

Parameters - MLM_COPY

param1 (ULONG)
Reserved.
0
Reserved value, 0.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_COPY

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

---

param1

param1 (ULONG)
Reserved.
0
Reserved value, 0.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

ulCopy

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

MLM_CUT

Topics - MLM_CUT

Select an item:

Main - MLM_CUT

Cause:

This message copies the text that forms the currently selected text to the clipboard and then deletes it from the MLE control. When the text type of the MLE is implicit, the characters affected may not necessarily be adjacent in the text buffer : the action is taken on the visually selected text which is continous on the screen.

Parameters:

```
param1
    ULONG param1 Reserved.

param2
    ULONG param2 Reserved.

returns
    ULONG ulCopy Number of bytes transferred, counted in CF_TEXT format.
```

Notes - MLM_CUT

The multi-line entry field control window procedure responds to this message by copying the selected text to the clipboard and then deleting it. The text is translated to standard clipboard format, which is the same as exporting with MLE_CFTEXT format.

The text is placed on the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64KB).

This may cause an overflow, see MLN_OVERFLOW.

Default Processing - MLM_CUT

The default window procedure takes no action on this message, other than to set *ulCopy* to 0.

Parameters - MLM_CUT

param1 (ULONG)
Reserved.
0
Reserved value, 0.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_CUT

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

---

param1

param1 (ULONG)
Reserved.
0
Reserved value, 0.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

ulCopy

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

MLM_DELETE

Topics - MLM_DELETE

Select an item:

Main - MLM_DELETE

Cause:

This message deletes text based on the VISUAL selection.

Parameters:

```
param1
    IPT   iptBegin  VISUAL Starting point of deletion.

param2
    ULONG ulDel     Number of bytes to delete.

returns
    ULONG ulSuccess Number of bytes successfully deleted.
```

Notes - MLM_DELETE

This message takes an insertion point and a length, and deletes that number of characters from the text. If the insertion point is -1, the selection is used and the effect is identical to the MLM_CLEAR message.

This may cause an overflow, see MLN_OVERFLOW.

Default Processing - MLM_DELETE

The default window procedure takes no action on this message, other than to set *ulSuccess* to 0.

Parameters - MLM_DELETE

iptBegin (IPT)
Visual starting point of deletion.

ulDel (ULONG)
Number of bytes to delete.

Return Values - MLM_DELETE

ulSuccess (ULONG)
Number of bytes successfully deleted.

---

iptBegin

iptBegin (IPT)
Visual starting point of deletion.

---

ulDel

ulDel (ULONG)
Number of bytes to delete.

---

ulSuccess

ulSuccess (ULONG)
Number of bytes successfully deleted.

MLM_EXPORT

Topics - MLM_EXPORT

Select an item:

Main - MLM_EXPORT

Cause:

This message exports VISUALLY selected text to a buffer.

Parameters:

```
param1
    PIPT   pBegin    Visual starting point.

param2
    PULONG pCopy     Number of bytes being exported.

returns
    ULONG  ulSuccess Number of bytes successfully exported.
```

Notes - MLM_EXPORT

This message takes a visual insertion point and length as parameters, and copies text, starting from that insertion point, into the buffer set by MLM_SETIMPORTEXPORT. Text is in the format set by MLM_FORMAT. If the insertion point is -1, the selection is used for both **pBegin** and **pCopy**.

On return, **pBegin** is updated to follow the last byte exported, and the number of bytes to be exported is decremented by the number actually exported. This is done to prepare those parameter values for the next export. The return value indicates the number of bytes actually put into the buffer. This number is less than, or equal to, the buffer size (see MLM_SETIMPORTEXPORT).

Note: All exports are done in full characters. Therefore, if either the length of the buffer or the number of bytes to be exported result in the last byte transferred being only half of a DBCS character, the MLE will *not* transfer that byte.

It returns the number of bytes placed in the export buffer.

Default Processing - MLM_EXPORT

The default window procedure takes no action on this message, other than to set *ulSuccess* to 0.

Parameters - MLM_EXPORT

pBegin (PIPT)
Visual starting point.

Updated to follow the last character exported.

pCopy (PULONG)
Number of bytes being exported.

Decremented by the number of bytes actually exported.

Return Values - MLM_EXPORT

ulSuccess (ULONG)
Number of bytes successfully exported.

---

pBegin

pBegin (PIPT)
Visual starting point.

Updated to follow the last character exported.

---

pCopy

pCopy (PULONG)
Number of bytes being exported.

Decremented by the number of bytes actually exported.

---

ulSuccess

ulSuccess (ULONG)
Number of bytes successfully exported.

MLM_IMPORT

Topics - MLM_IMPORT

Select an item:

MLM_INSERT

Topics - MLM_INSERT

Select an item:

Main - MLM_INSERT

Cause:

This message deletes the currently selected text and replaces it with a text string. When the text type of the MLE is implicit, the characters affected may not necessarily be adjacent in the text buffer : the action is taken on the visually selected text which is continous on the screen.

Parameters:

```
param1
    PSTRL pText   Null-terminated text string.

param2
    ULONG param2  Reserved.

returns
    ULONG ulCount  Number of bytes actually inserted.
```

Notes - MLM_INSERT

This message inserts the text string at the currently selected text, visually deleting that selection. Unlike in the English mode, deletion does not occur in the same manner as typing at the keyboard would. This is because typing Arabic in overwrite mode would replace characters in logical order not visual order.

The text string must be in CF_TEXT format (or one of the formats acceptable to MLM_IMPORT) and null-terminated. The line-break (CR LF, LF, and so on) is counted as one byte, regardless of the number of bytes occupied in the buffer, and the null terminator is not counted.

This interacts with the format rectangle and text limits, and a return of less than the full count can be the result. If so, a notification message is sent.

Default Processing - MLM_INSERT

The default window procedure takes no action on this message, other than to set *ulCount* to 0.

Parameters - MLM_INSERT

pText (PSTRL)
Null-terminated text string.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_INSERT

ulCount (ULONG)
Number of bytes actually inserted.

---

pText

pText (PSTRL)
Null-terminated text string.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

ulCount

ulCount (ULONG)
Number of bytes actually inserted.

MLM_LINEFROMCHAR

Topics - MLM_LINEFROMCHAR

Select an item:

Main - MLM_LINEFROMCHAR

Cause:

This message returns the line number corresponding to a given VISUAL insertion point.

Parameters:

```
param1
    IPT   iptFirst VISUAL insertion point of interest

param2
    ULONG param2   Reserved.

returns
    LONG  lLineNum Line number of insertion point.
```

Notes - MLM_LINEFROMCHAR

For any visual insertion point, the corresponding line number is returned. If the insertion point is -1, the number of the line containing the first insertion point of the visual selection is returned.

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

Default Processing - MLM_LINEFROMCHAR

The default window procedure takes no action on this message, other than to set *lLineNum* to 0.

Parameters - MLM_LINEFROMCHAR

iptFirst (IPT)
Visual insertion point of interest.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_LINEFROMCHAR

lLineNum (LONG)
Line number of visual insertion point.

---

iptFirst

iptFirst (IPT)
Visual insertion point of interest.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

lLineNum

lLineNum (LONG)
Line number of visual insertion point.

MLM_PASTE

Topics - MLM_PASTE

Select an item:

Main - MLM_PASTE

Cause:

This message replaces the text that forms the current VISUAL selection, with text from the clipboard. When the text type of the MLE is implicit, the characters affected may not necessarily be adjacent in the text buffer : the action is taken on the visually selected text which is continous on the screen.

Parameters:

```
param1
    ULONG param1 Reserved.

param2
    ULONG param2 Reserved.

returns
    ULONG ulCopy Number of bytes transferred, counted in CF_TEXT format.
```

Notes - MLM_PASTE

The multi-line entry field control window procedure responds to this message by replacing the selected text with text from the clipboard. The text is translated from standard clipboard format, which is the same as importing with MLE_CFTEXT format.

The text is assumed to be in the clipboard as a single contiguous data segment. This restricts the amount to the maximum segment size (64Kb).

This can cause an overflow, see MLN_OVERFLOW.

Default Processing - MLM_PASTE

The default window procedure takes no action on this message, other than to set *ulCopy* to 0.

Parameters - MLM_PASTE

param1 (ULONG)
Reserved.
0
Reserved value, 0.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_PASTE

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

---

param1

param1 (ULONG)
Reserved.
0
Reserved value, 0.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

ulCopy

ulCopy (ULONG)
Number of bytes transferred, counted in CF_TEXT format.

MLM_QUERYFORMATLINELENGTH

Topics - MLM_QUERYFORMATLINELENGTH

Select an item:

Main - MLM_QUERYFORMATLINELENGTH

Cause:

This message returns the number of bytes to end of line after formatting has been applied.

Parameters:

```
param1
    IPT   iptStart VISUAL insertion point to count from.

param2
    ULONG param2   Reserved.

returns
    IPT   iptLine  Count of bytes to end of line.
```

Notes - MLM_QUERYFORMATLINELENGTH

For any insertion point, the number of bytes between that insertion point and the end of the line is returned, after the current formatting is applied. This is done visually not logically. If the insertion point is -1, the cursor position is used. This message differs from MLM_QUERYLINELENGTH in that the byte count returned reflects the effects of the current formatting set by MLM_FORMAT.

Default Processing - MLM_QUERYFORMATLINELENGTH

The default window procedure takes no action on this message, other than to set *iptLine* to 0.

Parameters - MLM_QUERYFORMATLINELENGTH

iptStart (IPT)
Insertion point to count from.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_QUERYFORMATLINELENGTH

iptLine (IPT)
Count of bytes to end of line.

---

iptStart

iptStart (IPT)
Visual insertion point to count from.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

iptLine

iptLine (IPT)
Count of bytes to end of line.

MLM_QUERYFORMATTEXTLENGTH

Topics - MLM_QUERYFORMATTEXTLENGTH

Select an item:

Main - MLM_QUERYFORMATTEXTLENGTH

Cause:

This message returns the length of a specified range of characters after the current formatting has been applied.

Parameters:

```
param1
    IPT   iptStart VISUAL insertion point to start from.

param2
    ULONG ulScan   Number of characters to convert to bytes.

returns
    ULONG ulText   Count of bytes in text after formatting.
```

Notes - MLM_QUERYFORMATTEXTLENGTH

This message returns the length in bytes of a range of characters after the current formatting is applied. This is done visually not logically. This differs from MLM_QUERYTEXTLENGTH in that:

- A range of insertion points can be queried.
- The byte count returned reflects the effects of the current formatting set by MLM_FORMAT.

Default Processing - MLM_QUERYFORMATTEXTLENGTH

The default window procedure takes no action on this message, other than to set *ulText* to 0.

Parameters - MLM_QUERYFORMATTEXTLENGTH

iptStart (IPT)
Insertion point to start from.

ulScan (ULONG)
Number of characters to convert to bytes.
0xFFFFFFFF
Convert until end of line

other
Convert specified number of characters.

Return Values - MLM_QUERYFORMATTEXTLENGTH

ulText (ULONG)
Count of bytes in text after formatting.

---

iptStart

iptStart (IPT)
Insertion point to start from.

---

ulScan

ulScan (ULONG)
Number of characters to convert to bytes.
0xFFFFFFFF
Convert until end of line

other
Convert specified number of characters.

---

ulText

ulText (ULONG)
Count of bytes in text after formatting.

MLM_QUERYLINELENGTH

Topics - MLM_QUERYLINELENGTH

Select an item:

Main - MLM_QUERYLINELENGTH

Cause:

This message returns the number of bytes between a given insertion point and the end of line.

Parameters:

```
param1
    IPT   iptStart VISUAL insertion point to count from.

param2
    ULONG param2   Reserved.

returns
    IPT   iptLine  Count of bytes to end of line.
```

Notes - MLM_QUERYLINELENGTH

For any insertion point, the number of bytes between that insertion point and the end of the line is returned. The visual, not logical, insertion point is used. If the insertion point is -1, the cursor position is used. If the line contains a hard line-break, it is counted as one byte.

The term line means a line on the display after the application of word-wrap. It does not mean a line as defined by the CR LF line-break sequence.

Default Processing - MLM_QUERYLINELENGTH

The default window procedure takes no action on this message, other than to set *iptLine* to 0.

Parameters - MLM_QUERYLINELENGTH

iptStart (IPT)
Insertion point to count from.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_QUERYLINELENGTH

iptLine (IPT)
Count of bytes to end of line.

---

iptStart

iptStart (IPT)
Insertion point to count from.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

iptLine

iptLine (IPT)
Count of bytes to end of line.

MLM_QUERYSEL

Topics - MLM_QUERYSEL

Select an item:

Main - MLM_QUERYSEL

Cause:

This message returns the location of the VISUAL selection.

Parameters:

```
param1
    USHORT usQueryMode Query Mode.

param2
    ULONG  param2      Reserved.

returns
    SHORT  sMinSel     Minimum visual insertion point of selection.
    SHORT  sMaxSel     Maximum visual insertion point of selection.
    IPT    iptipt      Requested visual insertion point.
```

Notes - MLM_QUERYSEL

This message returns the location of the visual selection in several different forms. The insertion points lie between characters, and start at a zero origin before the first character in the MLE. Subtracting the minimum from the maximum gives the number of characters in the selection. *This is not necessarily the number of bytes of ASCII* . The line-break character is a CR LF (2 bytes) and all DBCS characters are 2 bytes. To determine the number of bytes, use MLM_QUERYFORMATTEXTLENGTH, being sure that the format choice set by MLM_FORMAT is set to what is used when the data is exported from the MLE (for example, MLE_CFTEXT for MLM_QUERYSELTEXT).

Note the following:

- If anchor point > cursor point, minimum point = cursor point and maximum point = anchor point.
- If anchor point < cursor point, minimum point = anchor point and maximum point = cursor point.

Default Processing - MLM_QUERYSEL

The default window procedure takes no action on this message, other than to set *reply* to 0.

Parameters - MLM_QUERYSEL

usQueryMode (USHORT)
Query Mode.
MLFQS_MINMAXSEL
Return both minimum and maximum visual points of selection in a format compatible with the EM_QUERYSEL message.

MLFQS_MINSEL
Return minimum visual insertion point of selection.

MLFQS_MAXSEL
Return maximum visual insertion point of selection.

MLFQS_ANCHORSEL
Return visual anchor point of selection.

MLFQS_CURSORSEL
Return visual cursor point of selection.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_QUERYSEL

sMinSel (SHORT)
Minimum visual insertion point of selection.

This value is rounded down to 65 535, if necessary.

sMaxSel (SHORT)
Maximum visual insertion point of selection.

This value is rounded down to 65 535 if necessary.
For **usQueryMode** = MLFQS_MINSEL, MLFQS_MAXSEL, MLFQS_ANCHORSEL, or MLFQS_CURSORSEL:

iptipt (IPT)
Requested visual insertion point.

---

usQueryMode

usQueryMode (USHORT)
Query Mode.
MLFQS_MINMAXSEL
Return both minimum and maximum visual points of selection in a format compatible with the EM_QUERYSEL message.

MLFQS_MINSEL
Return minimum visual insertion point of selection.

MLFQS_MAXSEL
Return maximum visual insertion point of selection.

MLFQS_ANCHORSEL
Return visual anchor point of selection.

MLFQS_CURSORSEL
Return visual cursor point of selection.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

sMinSel

sMinSel (SHORT)
Minimum visual insertion point of selection.

This value is rounded down to 65 535, if necessary.

---

sMaxSel

sMaxSel (SHORT)
Maximum visual insertion point of selection.

This value is rounded down to 65 535 if necessary.
For **usQueryMode** = MLFQS_MINSEL, MLFQS_MAXSEL, MLFQS_ANCHORSEL, or MLFQS_CURSORSEL:

---

iptipt

iptipt (IPT)
Requested visual insertion point.

MLM_QUERYSELTEXT

Topics - MLM_QUERYSELTEXT

Select an item:

Main - MLM_QUERYSELTEXT

Cause:

This message copies the currently VISUALLY selected text into a logical buffer.

Parameters:

```
param1
    PSTRL pBuff   Logical buffer for text string.

param2
    ULONG param2  Reserved.

returns
    ULONG ulCount Number of bytes to put into text string.
```

Notes - MLM_QUERYSELTEXT

This message copies the current VISUALLY selected text into the logical buffer pointed to by **pBuff**. The text string is null-terminated. The byte count includes the text in CF_TEXT format (CR LF) and the null terminator.

Default Processing - MLM_QUERYSELTEXT

The default window procedure takes no action on this message, other than to set *ulCount* to 0.

Parameters - MLM_QUERYSELTEXT

pBuff (PSTRL)
Logical buffer for text string.

param2 (ULONG)
Reserved.
0

Reserved value, 0.

Return Values - MLM_QUERYSELTEXT

ulCount (ULONG)
Number of bytes to put into text string.

---

pBuff

pBuff (PSTRL)
Logical buffer for text string.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

ulCount

ulCount (ULONG)
Number of bytes to put into text string.

MLM_SEARCH

Topics - MLM_SEARCH

Select an item:

Main - MLM_SEARCH

Cause:

This message searches for a specified text string.

Parameters:

```
param1
    ULONG           ulStyle  Style flags.

param2
    PMLE_SEARCHDATA pse      Search specification structure.

returns
    BOOL            fSuccess Success indicator.
```

Notes - MLM_SEARCH

This message searches the MLE text for a specified string, starting at a specified insertion point and continuing until the second specified insertion point has been reached, or the requested string has been matched.

When an MLM_SEARCH message is sent, the text is scanned starting with the character that follows the insertion point indicated in the **iptStart** field of the MLE_SEARCHDATA structure. The search proceeds until the point indicated in the **iptStop** field, until a match is found, or until TRUE is returned from MLN_SEARCHPAUSE notification (see WM_CONTROL (in Multiline Entry Fields)). If a negative value is specified for the **iptStart**, the current cursor point is used. If a negative value is specified for **iptStop**, the end of the text is used. If **iptStop**, is less than or equal to **iptStart**, after performing the two indicated substitutions, the search wraps from the end of the text to the beginning of the text.

If the MLFSEARCH_CASESENSITIVE option is specified, the bytes of the search string must exactly match those in the text. If MLFSEARCH_CASESENSITIVE is not specified, the WinUpperChar of the search string must match the WinUpperChar of the text.

When a match is found, the **iptStart** field of the search specification structure is set to indicate the insertion point immediately preceding the first character of the match, and the **cchFind** field is set to indicate the number of characters in the match. The cursor selection is not altered unless MLFSEARCH_SELECTMATCH is specified. If it is, an MLM_SETSEL is done with the anchor point at **iptStart** and the cursor at **iptStart** + **cchFind**.

While searching, the MLE occasionally sends an MLN_SEARCHPAUSE notification message. If the owner responds to this message with the value TRUE, the MLE stops the search. When a search is stopped from MLN_SEARCHPAUSE, **iptStart** is set to the point where the search terminated. If the response is FALSE, the search continues (see also the definition of MLN_SEARCHPAUSE). The interval at which MLN_SEARCHPAUSE notifications are sent is implementation-dependent, but must not exceed reasonable user-response thresholds, nor should it be so often as to introduce undue messaging overhead. Sending this notification every half second is a reasonable compromise.

When no match is found the **iptStart** value is unchanged.

If the application needs to continue the search, the proper way is to change the **iptStart** value to be the point following the string found, adjusting for any text changes done after the search that may have moved the relative location of the point.

Applications using this message are advised to change the system pointer to the wait icon (clock face) if it is expected that the search will take some time.

Default Processing - MLM_SEARCH

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

Parameters - MLM_SEARCH

ulStyle (ULONG)
Style flags.
MLFSEARCH_CASESENSITIVE
If set, only exact matches are considered a successful match. If not set, any case-combination of the correct characters in the correct sequence is considered a successful match.

MLFSEARCH_SELECTMATCH
If set, the MLE selects the text and scrolls it into view when found, just as if the application had sent an MLM_SETSEL message. This is not done if MLFSEARCH_CHANGEALL is also indicated.

MLFSEARCH_CHANGEALL
Using the MLE_SEARCHDATA structure specified in **pse**, all occurrences of **pchFind** are found, searching from **iptStart** to **iptStop**, and replacing them with **pchReplace**. If this style is selected, the **cchFound** field has no meaning, and the **iptStart** value points to the place where the search stopped, or is the same as **iptStop** because the search has not been stopped at any of the found strings. The current cursor location is not moved. However, any existing selection is deselected.

pse (PMLE_SEARCHDATA)
Search specification structure.

Return Values - MLM_SEARCH

fSuccess (BOOL)
Success indicator.
TRUE
The search was successful.

FALSE
The search was unsuccessful.

---

ulStyle

ulStyle (ULONG)
Style flags.
MLFSEARCH_CASESENSITIVE
If set, only exact matches are considered a successful match. If not set, any case-combination of the correct characters in the correct sequence is considered a successful match.

MLFSEARCH_SELECTMATCH
If set, the MLE selects the text and scrolls it into view when found, just as if the application had sent an MLM_SETSEL message. This is not done if MLFSEARCH_CHANGEALL is also indicated.

MLFSEARCH_CHANGEALL

Using the MLE_SEARCHDATA structure specified in **pse**, all occurrences of **pchFind** are found, searching from **iptStart** to **iptStop**, and replacing them with **pchReplace**. If this style is selected, the **cchFound** field has no meaning, and the **iptStart** value points to the place where the search stopped, or is the same as **iptStop** because the search has not been stopped at any of the found strings. The current cursor location is not moved. However, any existing selection is deselected.

---

pse

pse (PMLE_SEARCHDATA)
Search specification structure.

---

fSuccess

fSuccess (BOOL)
Success indicator.
TRUE
The search was successful.

FALSE
The search was unsuccessful.

MLM_SETFIRSTCHAR

Topics - MLM_SETFIRSTCHAR

Select an item:

Main - MLM_SETFIRSTCHAR

Cause:

This message sets the first visible character.

Parameters:

```
param1
    IPT   iptFVC   Visual insertion point to place in top left-hand corner.

param2
    ULONG param2   Reserved.

returns
    BOOL  fSuccess Success indicator.
```

Notes - MLM_SETFIRSTCHAR

This message scrolls the text to place the character following the insertion point into the upper left-hand corner of the window. If the insertion point specified is beyond the end of a line, or the end of the file, it is resolved in the same way as it is for a mouse click.

Default Processing - MLM_SETFIRSTCHAR

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

Parameters - MLM_SETFIRSTCHAR

iptFVC (IPT)
Visual insertion point to place in top left-hand corner.

param2 (ULONG)
Reserved.
0
Reserved value, 0.

Return Values - MLM_SETFIRSTCHAR

fSuccess (BOOL)
Success indicator.
TRUE
Successful completion

FALSE
An error occurred.

---

iptFVC

iptFVC (IPT)
Visual insertion point to place in top left-hand corner.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

fSuccess

fSuccess (BOOL)
Success indicator.
TRUE
Successful completion

FALSE
An error occurred.

MLM_SETSEL

Topics - MLM_SETSEL

Select an item:

Main - MLM_SETSEL

Cause:

This message sets a VISUAL selection.

Parameters:

```
param1
    IPT  iptAnchor Visual insertion point for new visual anchor point.

param2
    IPT  iptCursor Visual insertion point for new visual cursor point.

returns
    BOOL fSuccess  Success indicator.
```

Notes - MLM_SETSEL

This message sets the visual anchor and visual cursor points. The screen display is updated appropriately, ensuring that the cursor point is visible (which may involve scrolling). Note that the text cursor and inversion are not displayed if the MLE window does not have the input focus. A negative value for a point leaves that point alone.

Default Processing - MLM_SETSEL

The default window procedure takes no action on this message, other than to set *fSuccess* to FALSE.

Parameters - MLM_SETSEL

iptAnchor (IPT)
: Visual insertion point for new visual anchor point.

iptCursor (IPT)
: Visual insertion point for new visual cursor point.

Return Values - MLM_SETSEL

fSuccess (BOOL)
: Success indicator.

---

iptAnchor

iptAnchor (IPT)
Visual insertion point for new visual anchor point.

---

iptCursor

iptCursor (IPT)
Visual insertion point for new visual cursor point.

---

fSuccess

fSuccess (BOOL)
Success indicator.

Notebook control

Processing of Notebook control messages in the PM Bidirectional environment is the same as in the non-bidirectional PM environment. However, the Notebook control is sensitive to its Bidirectional attributes and handles formatting and presentation accordingly.

When the notebook control has right to left window orientation, the page arrows have their roles reversed: the left arrow turns the pages forward, whereas the right one turns them backwards. The text of tabs is converted according to their bidi attributes.

The WM_SETBIDIATTR dynamicaly modifies the bidi attributes (and as a result - the appearance) of the notebook control. It will also send appropriate messages to the user-defined page windows.

BKM_QUERYSTATUSLINEBIDIATTR

Topics - BKM_QUERYSTATUSLINEBIDIATTR

Select an item:

MAIN - BKM_QUERYSTATUSLINEBIDIATTR

Cause:

This message queries the bidi attribute of the status line text.

Parameters:

```
param1
    ULONG       ulPageId    Page ID.

param2
    PULONG      pBidiAttr     Pointer.

returns
    BOOL        fsuccess    Success indicator.
```

PARAMETERS - BKM_QUERYSTATUSLINEBIDIATTR

ulpageID (ULONG)
Page ID.

Page identifier of the page whose status line text is requested.

pBidiAttr (PULONG)
Pointer.

Pointer to a BIDIATTR data structure where the statusline bidi attributes are returned.

---

ulpageID

ulpageID (ULONG)
Page ID.

Page identifier of the page whose status line text is requested.

---

pBidiAttr

pBidiAttr (PULONG)
Pointer.

Pointer to a BIDIATTR data structure defining the status line text.

RETURN VALUES - BKM_QUERYSTATUSLINEBIDIATTR

fSuccess(BOOL)
Success indicator.
TRUE
Query of the notebook status line succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Query of the notebook status line succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - BKM_QUERYSTATUSLINEBIDIATTR

This is the main expl.of cm NOTES

DEFAULT PROCESSING - BKM_QUERYSTATUSLINEBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

BKM_QUERYTABTEXTBIDIATTR

Topics - BKM_QUERYTABTEXTBIDIATTR

Select an item:

MAIN - BKM_QUERYTABTEXTBIDIATTR

Cause:

This message queries the bidi attribute of the tab text.

Parameters:

```
param1
    ULONG    ulPageID    Page ID.

param2
    PULONG   pBidiAttr     Pointer.

returns
    BOOL     fsuccess    Success indicator.
```

PARAMETERS - BKM_QUERYTABTEXTBIDIATTRR

ulpageID (ULONG)
Page ID.

Page identifier of the page whose tab text is queried for Bidi attributes.

pBidiAttr (PULONG)
Pointer.

Pointer to a BIDIATTR data structure where the bidirectional attributes of the tab text are returned.

---

ulpageID

ulpageID (ULONG)
Page ID.

Page identifier of the page whose tab text is queried for bidirectional attributes.

---

pBidiAttr

pBidiAttr (PULONG)
Pointer.

Pointer to a BIDIATTR data structure where the bidirectional attributes of the tab text are returned.

RETURN VALUES - BKM_QUERYTABTEXTBIDIATTR

fSuccess(BOOL)
Success indicator.
TRUE
Query of the notebook tab text bidirectional attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Query of the notebook tab text bidirectional attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - BKM_QUERYTABTEXTBIDIATTR

The mnemonic specified should treat all Arabic shapes in the same way. They should normalize both the typed key and the character in the TAB text to their "base shapes". This processing should be the same as done in the menu control.

DEFAULT PROCESSING - BKM_QUERYTABTEXTBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

BKM_SETSTATUSLINEBIDIATTR

Topics - BKM_SETSTATUSLINEBIDIATTR

Select an item:

MAIN - BKM_SETSTATUSLINEBIDIATTR

Cause:

This message sets the bidi attribute for a specific status line.

Parameters:

```
param1
    ULONG           ulPageID       Page ID.

param2
    PBD_ATTR_MASK   pBD_Attr_Mask  Pointer.

returns
    BOOL            fsuccess       Success indicator.
```

PARAMETERS - BKM_SETSTATUSLINEBIDIATTR

ulpageID (ULONG)
Page ID.

Page identifier of the page with which to associate the text string.

pBD_Attr_Mask (PBD_ATTR_MASK)
Pointer.

Pointer to the BD_ATTR_MASK data structure defining the bidirectional attributes to be set.

---

ulpageID

ulpageID (ULONG)
Page ID.

Page identifier of the page whose status line text is assigned with the new bidirectional attributes.

---

pBD_Attr_Mask

pBD_Attr_Mask (PBD_ATTR_MASK)
Pointer.

Pointer to the BD_ATTR_MASK data structure containing the bidirectional attributes.

RETURN VALUES - BKM_SETSTATUSLINEBIDIATTR

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the notebook status line bidirectional attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the notebook status line bidirectional attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - BKM_SETSTATUSLINEBIDIATTR

DEFAULT PROCESSING -

BKM_SETSTATUSLINEBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

BKM_SETTABTEXTBIDIATTR

Topics - BKM_SETTABTEXTBIDIATTR

Select an item:

MAIN - BKM_SETTABTEXTBIDIATTR

Cause:

This message sets the bidi attribute for a specific tab text.

Parameters:

```
param1
    ULONG           ulPageID       Page ID.

param2
    PBD_ATTR_MASK   pBD_Attr_Mask  Pointer.

returns
    BOOL            fsuccess       Success indicator.
```

PARAMETERS - BKM_SETTABTEXTBIDIATTR

ulpageID (ULONG)
Page ID.

Page identifier of the page whose tab text is assigned with new bidirectional attributes.

pBD_Attr_Mask (PBD_ATTR_MASK)
Pointer.

Pointer to the BD_ATTR_MASK data structure containing the new bidirectional attributes.

---

ulpageID

ulpageID (ULONG)
Page ID.

Page identifier of the page whose tab text is assigned with the new bidirectional attributes.

---

pBD_Attr_Mask

pBD_Attr_Mask (PBD_ATTR_MASK)
Pointer.

Pointer to the BD_ATTR_MASK data structure containte the new bidirectional attributes.

RETURN VALUES - BKM_SETTABTEXTBIDIATTR

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the notebook tab text bidirectional attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the notebook tab text bidirectional attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - BKM_SETTABTEXTBIDIATTR

The mnemonic specified should treat all Arabic shapes in the same way. They should normalize both the typed key and the character in the TAB text to their "base shapes". This processing should be the same as done in the menu control.

DEFAULT PROCESSING - BKM_SETTABTEXTBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

BKM_QUERYPAGEINFO

The mp2 parameter of this message points to a BOOKPAGEINFO data structure. The pBidiInfo field in this data structure points to the BOOKPAGEBIDIINFO structure.

The "fl" field of the BOOKPAGEINFO structure must have the BFA_BIDIINFO flag set to indicate that Bidi Attributes of the MajorTab/MinorTab/Status line text (as indicated by the other flags in the "fl" field) are queried.

BKM_SETPAGEINFO

The mp2 parameter of this message points to a BOOKPAGEINFO data structure. The pBidiInfo field in this data structure points to the BOOKPAGEBIDIINFO structure.

The "fl" field of the BOOKPAGEINFO structure must have the BFA_BIDIINFO flag set to indicate that Bidi Attributes contained in the BOOKPAGEBIDIINFO structure are to be set for the Major Tab/Minor Tab/Status Line as indicated by the other flags set in this field.

Value Set Control

Processing of the Value-Set control messages in the PM Bidirectional environment is the same as in the non-bidirectional PM environemnt. However, the Value-Set control is sensitive to its Bidirectional attributes and handles formatting and presentation accordingly.

VM_QUERYITEMBIDIATTR

Topics - VM_QUERYITEMBIDIATTR

Select an item:

MAIN - VM_QUERYITEMBIDIATTR

Cause:

This message queries the bidi attributes for each text item in an application.

Parameters:

```
param1
     USHORT          usRow           Row index.
     USHORT          usColumn        Column index.

param2
     PULONG          pBidiAttr       Pointer.

returns
     BOOL            fsuccess        Success indicator.
```

PARAMETERS - VM_QUERYITEMBIDIATTR

usRow (USHORT)
Row index.

Row index of the items to be queried. Rows have a value from 1 to **usRowCount** field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

usColumn (USHORT)
Column index.

Column index of the items to be queried. Columns have a value from 1 to **usColumnCount** field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

pBidiAttr(PULONG)
Pointer.

pointer to the BIDIATTR structure.

PARAMETERS - VM_QUERYITEMBIDIATTR

usRow (USHORT)
Row index.

Row index of the items to be queried. Rows have a value from 1 to **usRowCount** field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

PARAMETERS - VM_QUERYITEMBIDIATTR

usColumn (USHORT)
Column index.

Column index of the items to be queried. Columns have a value from 1 to **usColumnCount** field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

---

param2

pBidiAttr(PULONG)
Pointer.

pointer to the BIDIATTR structure.

RETURN VALUES - VM_QUERYITEMBIDIATTR

fSuccess(BOOL)
Success indicator.
TRUE
Query of the item bidi attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Query of the item bidi attributes succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - VM_QUERYITEMBIDIATTR

If item bidi attribute is not set (BDA_INIT is 0) the behavior is defined (inherited) from the bidi attributes of the valueset.

DEFAULT PROCESSING - VM_QUERYITEMBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

VM_SETITEMBIDIATTR

Topics - VM_SETITEMBIDIATTR

Select an item:

MAIN - VM_SETITEMBIDIATTR

Cause:

This message sets the bidi attributes for each text item in an application.

Parameters:

```
param1
     USHORT    usRow     Row index.
     USHORT    usColumn  Column index.

param2
     ULONG     param2    Reserved.

returns
     BOOL      fsuccess  Success indicator.
```

PARAMETERS - VM_SETITEMBIDIATTR

usRow (USHORT)
Row index.

Row index of the value set item for which information is being specified. Rows have a value from 1 to **usRowCount** field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

usColumn (USHORT)
Column index.

Column index of the value set item for which information is being specified. Columns have a value from 1 to **usColumnCount** field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

param2 (ULONG)
Reserved.

0
Reserved value, 0.

PARAMETERS - VM_SETITEMBIDIATTR

usRow (USHORT)
Row index.

Row index of the value set item for which information is being specified. Rows have a value from 1 to **usRowCount** field. This value, which is the total number of rows in the value set, is specified in the VSCDATA data structure when the value set control is created.

PARAMETERS - VM_SETITEMBIDIATTR

usColumn (USHORT)
Column index.

Column index of the value set item for which information is being specified. Columns have a value from 1 to **usColumnCount** field. This value, which is the total number of columns in the value set, is specified in the VSCDATA data structure when the value set control is created.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

RETURN VALUES - VM_SETITEMBIDIATTR

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the bidi attribute succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Setting of the bidi attribute succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - VM_SETITEMBIDIATTR

If item bidi attribute is not set (BDA_INIT is 0) the behavior is defined (inherited) from the bidi attributes of the valueset.

DEFAULT PROCESSING - VM_SETITEMBIDIATTR

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

General Bidi Window Messages

Processing of general window messages in the PM Bidirectional environment is the same as in the non-bidirectional PM environemnt.

WM_LANG_OPTIONS_DIALOG

Topics - WM_LANG_OPTIONS_DIALOG

Select an item:

MAIN - WM_LANG_OPTIONS_DIALOG

Cause:

This message is posted to the viewer to notify it that the language options configuration dialog for a certain window should be invoked.

Parameters:

```
param1
     HWND       Whwnd       Window handle.

param2
     ULONG      param2      Reserved.

returns
     BOOL       fsuccess    Success indicator.
```

PARAMETERS - WM_LANG_OPTIONS_DIALOG

Whwnd (HWND)
Window handle.

This contains the window handle of the window whose language options are to be configured.

param2 (ULONG)
Reserved.

1
Reserved value, 1, for bidi.

Whwnd

Whwnd (HWND)
This contains the window handle of the window whose language options are to be configured.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

RETURN VALUES - WM_LANG_OPTIONS_DIALOG

fSuccess(BOOL)
Success indicator.
TRUE
Changes in the bidi viewer succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Changes in the bidi viewer succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - WM_LANG_OPTIONS_DIALOG

This message may be posted when the user wishes to invoke the languages option dialog using bidi key combinations.

DEFAULT PROCESSING - WM_LANG_OPTIONS_DIALOG

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

WM_LANGVIEWINFOCHANGED

Topics - WM_LANGVIEWINFOCHANGED

Select an item:

MAIN - WM_LANGVIEWINFOCHANGED

Cause:

This message is sent by the system to notify the active language viewer window of the changes in the 'Bidi stat'.

Parameters:

```
param1
     ULONG      ulnotify    Type definition.

param2
     ULONG      param2      Reserved.

returns
     BOOL       fsuccess    Success indicator.
```

PARAMETERS - WM_LANGVIEWINFOCHANGED

ulnotify (ULONG)
Type definition.

This contains the value of the type of event that has taken place and which is notified to the viewer.

param2 (ULONG)
Reserved.

0
Reserved value, 0.

---

param2

ulnotify (ULONG)
This contains the value of the type of event that has taken place and which is notified to the viewer.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

RETURN VALUES - WM_LANGVIEWINFOCHANGED

fSuccess(BOOL)
Success indicator.
TRUE
Changes in the bidi viewer succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Changes in the bidi viewer succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - WM_LANGVIEWINFOCHANGED

The first parameter of this message defines the type of event that has taken place and which is notified to the bidi viewer.

DEFAULT PROCESSING - WM_LANGVIEWINFOCHANGED

The Default window procedure does not expect to receive this message and therefore takes no action on it other than to return FALSE

WM_QUERYBIDIATTR

Topics - WM_QUERYBIDIATTR

Select an item:

MAIN - WM_QUERYBIDIATTR

Cause:

This message is sent to a window procedure to query its bidi attributes.

Parameters:

```
param1
     ULONG     param1      Reserved.

param2
     ULONG     param2      Reserved.

returns
     BOOL      fsuccess    Success indicator.
```

PARAMETERS - WM_QUERYBIDIATTR

param1 (ULONG)
Reserved.

0
Reserved value, 0.

param2 (ULONG)
Reserved.

0
Reserved value, 0.

RETURN VALUES - WM_QUERYBIDIATTR

fSuccess(BOOL)
Success indicator.
TRUE
Query of the bidi attribute succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

param1

param1 (ULONG)
Reserved.
0
Reserved value, 0.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Query of the bidi attribute succeeded.

FALSE

An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - WM_QUERYBIDIATTR

This message is sent to a window procedure to query it's bidi attributes. This is done as part of WinQueryLangInfo processing.

DEFAULT PROCESSING - WM_QUERYBIDIATTR

The Default window procedure returns the window bidi attributes, whose values are contained in the BIDIATTR.

WM_QUERYBIDISTAT

Topics - WM_QUERYBIDISTAT

Select an item:

MAIN - WM_QUERYBIDISTAT

Cause:

This message is sent to a window procedure to query its bidi status.

Parameters:

```
param1
     ULONG       param1      Reserved.

param2
     ULONG       param2      Reserved.

returns
```

```
BOOL        fsuccess    Success indicator.
```

PARAMETERS - WM_QUERYBIDISTAT

param1 (ULONG)
Reserved.

0
Reserved value, 0.

param2 (ULONG)
Reserved.

0
Reserved value, 0.

RETURN VALUES - WM_QUERYBIDISTAT

fSuccess(BOOL)
Success indicator.
TRUE
Query of the bidi status succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

---

param1

param1 (ULONG)
Reserved.
0
Reserved value, 0.

---

param2

param2 (ULONG)
Reserved.
0
Reserved value, 0.

---

fsuccess

fSuccess(BOOL)
Success indicator.
TRUE
Query of the bidi status succeeded.

FALSE
An error occurred. The WinGetLastError function may return the following error:

- PMERR_INVALID_PARAMETERS

NOTES - WM_QUERYBIDISTAT

This message is sent to a window procedure to query it's bidi status (these values are contained in the BIDI_STAT). This is done as part of WinQueryLangInfo processing.

DEFAULT PROCESSING - WM_QUERYBIDISTAT

The Default window procedure returns the window Bidi status (these values are contained in the BIDI_STAT).

WM_SETBIDIATTR

Topics - WM_SETBIDIATTR

Select an item:

MAIN - WM_SETBIDIATTR

Cause:

This message is sent to a window procedure to notify it that the bidi attributes of that window are about to be changed.

Parameters:

```
param1
    ULONG   ulData     Attribute data.

param2
    ULONG   ulMask     Value mask.

returns
    ULONG   fresult    Result value.
```

PARAMETERS - WM_SETBIDIATTR

ulData (ULONG)
Attribute data.

The value of the specified fields to be set in the Bidi attributes. These values are contained in the BIDIATTR.

ulMask (ULONG)
Value Mask.

A mask value that is used to indicate which fields, in the Bidi attributes structure have to be set. Only the bits that are set in ulMask (see BIDIATTRM), are enabled for update by the values in ulData.

---

param1

ulData (ULONG)
Attribute data.

The value of the specified fields to be set in the Bidi attributes. These values are contained in the BIDIATTR.

---

param2

ulMask (ULONG)
Value mask.

A mask value that is used to indicate which fields, in the Bidi attributes structure have to be set. Only the bits that are set in ulMask (see BIDIATTRM), are enabled for update by the values in ulData.

RETURN VALUES - WM_SETBIDIATTR

fresult(ULONG)
Result indicator.
SBI_MSG_NOT_PROCESSED:
This message notifies the sender that the message has NOT been processed by the window procedure. This is returned from the default window procedure.

SBI_MSG_PROCESSED:
This message notifies the sender that the message has been processed by the window procedure.

SBI_MSG_PROCESSED_SELF:
This message notifies the sender that the window procedure has processed the message for itself, but it leaves it for the sender to determine the processing that has to be done for children.

---

fresult

fresult(ULONG)
Result indicator.
SBI_MSG_NOT_PROCESSED:
This message notifies the sender that the message has NOT been processed by the window procedure. This is returned from the default window procedure.

SBI_MSG_PROCESSED:
This message notifies the sender that the message has been processed by the window procedure.

SBI_MSG_PROCESSED_SELF:
This message notifies the sender that the window procedure has processed the message for itself, but it leaves it for the sender to determine the processing that has to be done for children.

NOTES - WM_SETBIDIATTR

The message gives the window procedure a chance to ignore the request, set the attributes itself, or let the system default processing handle the operation for it.

It is optionaly sent to a window procedure when the bidi attributes of this window are about to be changed (the value is the BIDIATTR). This is done as part of WinQueryLangInfo processing.

DEFAULT PROCESSING - WM_SETBIDIATTR

When the WinSetLangInfo function sends this message, the default window returns SBI_MSG_NOT_PROCESSED which causes WinQueryLangInfo to set the Bidi Attributes by itself (the value is contained in the BIDIATTR).

WM_SETBIDISTAT

Topics - WM_SETBIDISTAT

Select an item:

MAIN - WM_SETBIDISTAT

Cause:

This message is sent to a window procedure to notify it that the bidi status of that window is about to be changed.

Parameters:

```
param1
    ULONG    ulData     Attribute data.

param2
    ULONG    ulMask     Value mask.

returns
    ULONG    fresult    Result value.
```

PARAMETERS - WM_SETBIDISTAT

ulData (ULONG)
Attribute data.

The value of the specified fields to be set in the Bidi status. These values are contained in the BIDISTAT.

ulMask (ULONG)
Value Mask.

A mask value that is used to indicate which fields, in the Bidi status structure have to be set. Only the bits that are set in ulMask (see BIDISTATM), are enabled for update by the values in ulData.

---

param1

ulData (ULONG)
Attribute data.

The value of the specified fields to be set in the Bidi status. These values are contained in the BIDISTAT.

---

param1

ulMask (ULONG)
Value mask.

A mask value that is used to indicate which fields, in the Bidi status structure have to be set. Only the bits that are set in ulMask (see BIDISTATM), are enabled for update by the values in ulData.

RETURN VALUES - WM_SETBIDISTAT

fresult(ULONG)
Result indicator.
SBI_MSG_NOT_PROCESSED:
This message notifies the sender that the message has NOT been processed by the window procedure. This is returned from the default window procedure.

SBI_MSG_PROCESSED:
This message notifies the sender that the message has been processed by the window procedure.

SBI_MSG_PROCESSED_SELF:
This message notifies the sender that the window procedure has processed the message for itself, but it leaves it for the sender to determine the processing that has to be done for children.

---

fresult

fresult(ULONG)
Result indicator.
SBI_MSG_NOT_PROCESSED:
This message notifies the sender that the message has NOT been processed by the window procedure. This is returned from the default window procedure.

SBI_MSG_PROCESSED:
This message notifies the sender that the message has been processed by the window procedure.

SBI_MSG_PROCESSED_SELF:
This message notifies the sender that the window procedure has processed the message for itself, but it leaves it for the sender to determine the processing that has to be done for children.

NOTES - WM_SETBIDISTAT

The message gives the window procedure a chance to ignore the request, set the status itself, or let the system default processing handle the operation for it.

It is optionaly sent to a window procedure when the bidi status of this window are about to be changed (the value is contained in the

BIDI_STAT). This is done as part of WinQueryLangInfo processing.

DEFAULT PROCESSING - WM_SETBIDISTAT

When the WinSetLangInfo function sends this message, the default window procedure returns SBI_MSG_NOT_PROCESSED which causes WinSetLangInfo to set the Bidi Status by itself. (the value is contained in the BIDI_STAT).

Bidirectional Support Window Functions

This section contains an alphabetical list of the functions which are available for controlling windows. These functions enable an application to create, size, move, and control windows and their contents in the *Bidirectional* programming environment.

See the *Presentation Manager Programming Guide and Reference* for details on commom programming techniques for managing the window environment, in general.

Win_BD_ClassifyCodepage

Topics - Win_BD_ClassifyCodepage

Select an item:

- Function Syntax
- Parameters
- Return Values
- Example

Syntax - Win_BD_ClassifyCodepage

```
/*****************************************/
/* This function returns a flag to classify */
/* the given codepage.                   */
/*****************************************/

#define INCL_PMBIDI
#include <os2.h>

ULONG     codepage;     /* Codepage value */
ULONG     cp;           /* Return value   */

   cp  =  Win_BD_ClassifyCodepage ( codepage );
```

Parameters - Win_BD_ClassifyCodepage

codepage (ULONG) - input
code page.

The value of the code page to be classified.

Return Values - Win_BD_ClassifyCodepage

cp (ULONG) - return
Code page indicator.

OL
A none Bidi code page.

CP_IS_BIDI
The code page is bidi. The following are possible values:

CP_IS_ARABIC
The code page is Arabic.
CP_IS_ARABIC2
The code page is Arabic 2.

Errors - Win_BD_ClassifyCodepage

Notes - Win_BD_ClassifyCodepage

Example - Win_BD_ClassifyCodepage

to be added

Related Functions - Win_BD_ClassifyCodepage

Related Functions

- There are no related functions.

Win_BD_IsStringAllBidi

Topics - Win_BD_IsStringAllBidi

Select an item:

Syntax - Win_BD_IsStringAllBidi

```
/*****************************************/
/* This function checks if a string is an */
/* 'AllBidi' string.                      */
/*****************************************/

#define INCL_PMBIDI
#include <os2.h>

PCHAR     pch;          /* Input string */
ULONG     ulMaxlen;     /* Number of characters */
ULONG     allBidi;      /* Return value   */

   allBidi  =  Win_BD_IsStringAllBidi ( pch, ulMaxlen );
```

Parameters - Win_BD_IsStringAllBidi

pch (PCHAR) - input
Input string.

The input string to be checked.

ulMaxlen (ULONG) - input

Number of characters.

The number of characters in the string to be checked.

Return Values - Win_BD_IsStringAllBidi

allBidi (ULONG) - return
All Bidi indicator.

TRUE
The string contains one or more National Language Characters.

FALSE
The string does not contain any National Language characters.

Errors - Win_BD_IsStringAllBidi

Notes - Win_BD_IsStringAllBidi

The string is an "allBidi" string if all the characters in the string are only Bidi characters (i.e. no LATIN characters)

Example - Win_BD_IsStringAllBidi

to be added

Related Functions - Win_BD_IsStringAllBidi

Related Functions

- Win_BD_IsStringBidi

Win_BD_IsStringBidi

Topics - Win_BD_IsStringBidi

Select an item:

Syntax - Win_BD_IsStringBidi

```
/*****************************************/
/* This function checks if a string is a  */
/* Bidi string.                           */
/*****************************************/

#define INCL_PMBIDI
#include <os2.h>

PCHAR     pch;          /* Input string */
ULONG     ulMaxlen;     /* Number of characters */
ULONG     Bidistr;      /* Return value   */

   Bidistr  =  Win_BD_IsStringBidi ( pch, ulMaxlen );
```

Parameters - Win_BD_IsStringBidi

pch (PCHAR) - input
Input string.

The input string to be checked.

ulMaxlen (ULONG) - input
Number of characters.

The number of characters in the string to be checked.

Return Values - Win_BD_IsStringBidi

Bidistr(BOOL) - return
Bidi string indicator.

TRUE
String contains ONLY National Language Characters.

FALSE
String contains some LATIN characters.

Errors - Win_BD_IsStringBidi

Notes - Win_BD_IsStringBidi

A Bidi string is a string that contains at least one of the characters that are NATIONAL LANGUAGE characters in the appropriate codepage.

Example - Win_BD_IsStringBidi

to be added

Related Functions - Win_BD_IsStringBidi

Related Functions

- Win_BD_IsStringAllBidi

WinQueryKbdLayer

Topics - WinQueryKbdLayer

Select an item:

Syntax - WinQueryKbdLayer

```
/*************************************/
/* This function queries the current    */
/* or system keyboard layer.            */
/*************************************/

#define INCL_PMBIDI
#include <os2.h>

HWND      hwnd;         /* Window handle */
ULONG     ulLayerID;    /* Return value  */

      ulLayerID =  WinQueryKbdLayer( hwnd );
```

Parameters - WinQueryKbdLayer

hwnd (HWND) - input
Window handle.

The window handle of the window for which the keyboard layer is queried, or HWND_DESKTOP.

Return Values - WinQueryKbdLayer

ulLayerID(ULONG) - return
Result indicator.

Returns the value of the current keyboard layer.

KBDLAYER_0
The default/base layout used for the codepage.

KBDLAYER_1
The first country/language keyboard layout.

Errors - WinQueryKbdLayer

Notes - WinQueryKbdLayer

The current keyboard layer queried is the one maintained for the window.

Example - WinQueryKbdLayer

This example queries the keyboard layer and draws a message box saying whether it was national language (Arabic/Hebrew) or Latin.

```
#define INCL_PMBIDI
#define INCL_WINDIALOGS
#include <OS2.H>
#include <PMBIDI.H>

ULONG QueryKbdLayer(HWND hwnd)
 {
 ULONG KbdLayer;

    KbdLayer = WinQueryKbdLayer (hwnd); /* Window handle of the window whose */
                                        /* keyboard layer is queried         */
    if (KbdLayer)
           WinMessageBox (HWND_DESKTOP,
                          hwnd,
                          "The keyboard layer is National",
                          "Result",
                           0,
                           MB_OK);

    else
           WinMessageBox (HWND_DESKTOP,
                          hwnd,
                          "The keyboard layer is Latin",
                          "Result",
                           0,
                           MB_OK);

 }
```

Related Functions - WinQueryKbdLayer

Related Functions

- WinSetKbdLayer

WinQueryLangInfo

Topics - WinQueryLangInfo

Select an item:

Syntax - WinQueryLangInfo

```
/*****************************************/
/* This function returns the Bidirectional */
/* information of the data specified by    */
/* ulEffect.                               */
/*****************************************/

#define INCL_PMBIDI
#include <os2.h>

HWND      hwnd;         /* Window handle */
ULONG     ulEffect;     /* Specifies type of Bidi information to be queried */
ULONG     ulFlags;      /* Flag to determine optional behavior */
ULONG     ulReserved;   /* Reserved and must be zero */
ULONG     uldata;       /* Return value */

     ulData  =  WinQueryLangInfo ( hwnd, ulEffect, ulFlags, ulReserved );
```

Parameters - WinQueryLangInfo

hwnd (HWND) - input
Window handle.

Handle of the window to be queried. This parameter should be NULLHANDLE when the information being queried is not window-related.

ulEffect (ULONG) - input
Specifies type of Bidi information to be queried.

It determines which type of Bidi-information is queried Depending on its value.

LI_BD_CLIP_ATTR
Clipboard bidirectional attributes.

LI_BD_CLIP_CONV_ATTR
The clipboard conversion bidirectional attributes.

LI_BD_PROCESS_ATTR
The process bidirectional attributes.

LI_BD_WND_ATTR
The window Bidi attributes.

LI_BD_WND_STAT
The window Bidi status.

ulFlags (ULONG) - input
Flag to determine optional behavior.

This is a word of bit flags that determines optional behavior of this function. Possible values are:

LIF_NO_SENDMSG
: No message is sent to the window to inform it of change. This value is valid only when the operation is related to windows.

ulReserved (ULONG)
: Reserved.

Is reserved and must be Zero.

Return Values - WinQueryLangInfo

ulData (ULONG) - return
: Bidirectional value indicator.

BIDIATTR
: Bidi attribute information.

BIDISTAT
: Bidi status information.

NULL
: Query of the bidi attributes was unsuccessful.

Errors - WinQueryLangInfo

Notes - WinQueryLangInfo

The interpertation of this function is dependent on the value of ulEffect. This function is considered "window related" when ulEffect is LI_BD_WND_ATTR or LI_BD_WND_STAT.

When ulEffect is not window related then the value of hwnd must be NULL.

On the other hand, when ulEffect is window related, the function optionally sends a WM_QUERYBIDIATTR or WM_QUERYBIDISTAT message to the window procedure whose information is being queried. The window procedure which is queried for bidi information can take some action and/or perform any additional processing that is required at this time. When LIF_NO_SENDMSG flag is specified no window message is sent by the function. This can be used by the window procedure itself when it queries its own window-related bidirectional information (so that it does not receive a recursive message).

It is recommended that the LIF_NO_SENDMSG flag is not turned on when the function is called from outside the window procedure whose attributes are being queried. Applications should generally use the default (send the message to the window procedure), so that the bidi aware window procedures can take the appropriate actions when the request to query their bidi attributes is made.

Example - WinQueryLangInfo

This example queries the Staus bits of a window. If these indicated the characters to be FINAL, a message is printed saying so else it says

that they are not final characters

```
#define INCL_PMBIDI
#include <OS2.H>
#include <PMBIDI.H>


VOID QueryLangInfo(HWND hwnd)
 {
 ULONG  BidiStat;

    BidiStat = WinQueryLangInfo(hwnd,        /* Window handle of the window whose      */
                                             /* status is queried                      */
                        LI_BD_WND_STAT,      /* The window status is queried           */
                        LIF_NO_SENDMSG,
                        0L);                 /* Reserved to be 0L                      */

    if ( QUERY_BD_VALUE (BidiStat,BDSM_HKFLAGS) == BDS_HKFLAG_PUSH)
        WinMessageBox (HWND_DESKTOP,
                       hwnd,
                       "Push Hotkey is disabled",
                       "Result",
                        0,
                        MB_OK);
    else
        WinMessageBox (HWND_DESKTOP,
                       hwnd,
                       "Push Hotkey is enabled".
                       "Result",
                        0,
                        MB_OK);

 }
```

Related Functions - WinQueryLangInfo

Related Functions

- WinSetLangInfo

WinQueryLangViewer

Topics - WinQueryLangViewer

Select an item:

Syntax - WinQueryLangViewer

```
/*****************************************/
/*  This function queries the current      */
/*  'Language Viewer' window, registered   */
/*  by the system for a specific codepage. */
/*****************************************/

#define INCL_PMBIDI
#include <os2.h>

HAB       hab;          /* Anchor block    */
ULONG     Codepage;     /* Codepage value  */
HWND      hwnd;         /* Return value    */

      hwnd  =  WinQueryLangViewer( hab, Codepage );
```

Parameters - WinQueryLangViewer

hab (HAB) - anchor block
Anchor block.

The anchor block handle of the application.

Codepage (ULONG) - input
Window codepage.

The codepage for which the "Language Viewer" window is queried. The following codepage values are supported:

864	Arabic PC.
862	Hebrew PC.

Return Values - WinQueryLangViewer

hwnd (HWND) - return
Result indicator.

Returns the handle of the current language viewer.

NULLHANDLE
No language viewer is registered for this codepage.

Other
Handle of the current language viewer.

Errors - WinQueryLangViewer

Notes - WinQueryLangViewer

None.

Example - WinQueryLangViewer

This example queries the Language Viewer window handle registered currently in the system for the currently used code page.

```
#define INCL_PMBIDI
#include <OS2.H>
#include <PMBIDI.H>

HAB    hab;
HWND   hwnd;
HMQ    hmq;
ULONG  cp;


VOID QueryLanguageViewerWindow(VOID)
 {
    cp = WinQueryCp(hmq);                  /* Query current code page                */

    if (hwnd = WinQueryLangViewer(hab,     /* Anchor block handle of the application */
                                           /* that registered the Language Viewer    */
                          cp))             /* Current code page                      */

         WinMessageBox (HWND_DESKTOP,
                        hwnd,
                        "The Language Viewer handle was queried correctly",
                        "Result",
                         0,
                         MB_OK);
    else
         WinMessageBox (HWND_DESKTOP,
                        hwnd,
                        "No Language Viewer has been registered for this code page",
                        "Error",
                         0,
                         MB_OK);

 }
```

Related Functions - WinQueryLangViewer

Related Functions

- WinSetLangViewer

WinSetKbdLayer

Topics - WinSetKbdLayer

Select an item:

Syntax - WinSetKbdLayer

```
/*************************************/
/* This function sets the keyboard      */
/* layer.                               */
/*************************************/

#define INCL_PMBIDI
#include <os2.h>

HWND      hwnd;         /* Window handle */
ULONG     ulLayerID;    /* Specifies the keyboard layer */
ULONG     flFlags;      /* impact flages */
BOOL      fsuccess;     /* Return value  */

   fsuccess  =  WinSetKbdLayer  ( hwnd, ulLayerID, flFlags);
```

Parameters - WinSetKbdLayer

hwnd (HWND) - input
Window handle.

Window handle of the window for which the keyboard layer is set, or HWND_DESKTOP.

ulLayerID (ULONG) - input
Keyboard layer.

ID of the keyboard layer being set.

KBDLAYER_0
This corresponds to the default/base layout used for the codepage.

KBDLAYER_1
This corresponds to the first country/language keyboard layout. Usually, this maps to the specific language (e.g. Arabic or Hebrew layout)

flFlags (ULONG) - input

Impact flags.

Flags which impact the way WinSetKbdLayer works.

bit 0 = 0:
Post the WM_KBDLAYERCHANGED message.

bit 0 = 1:
Do not post the WM_KBDLAYERCHANGED message.

Return Values - WinSetKbdLayer

fSuccess (BOOL) - return
Success indicator.

TRUE
Setting of the keyboard layer succeeded.

FALSE
An error occurred. The WinGetLAstError function may return the following errors:

- PMERR_INVALID_HWND
- PMERR_INVALID_VALUE

Errors - WinSetKbdLayer

Notes - WinSetKbdLayer

The system maintains one global keyboard layer variable. This variable is updated according to the layer information of the window that has the keyboard focus.

WinSetKbdLayer saves the keyboard layer specified as an input parameter in an area maintained by the system on behalf of the window. If the window has the focus, the system keyboard layer is also set.

When the value of hwnd is HWND_DESKTOP, only the system keyboard layer is updated.

Example - WinSetKbdLayer

This example sets the keyboard layer to National and posts the message WM_KBDLAYERCHANGED through setting the impact flags bit to zero.

```
#define INCL_PMBIDI
#include <OS2.H>
#include <PMBIDI.H>


VOID SetKbdLayer(HWND hwnd)
```

```
{
   if (WinSetKbdLayer(hwnd,
             KL_NATIONAL,            /* First country/language layout       */
             SKLF_SENDMSG))          /* Post the WM_KBDLAYERCHANGED message */

        WinMessageBox (HWND_DESKTOP,
                       hwnd,
                       "Setting keyboard layer succeeded",
                       "Result",
                        0,
                        MB_OK);
   else
        WinMessageBox (HWND_DESKTOP,
                       hwnd,
                       "Setting keyboard layer failed",
                       "Error",
                        0,
                        MB_OK);

}
```

Related Functions - WinSetKbdLayer

Related Functions

- WinQueryKbdLayer

WinSetLangInfo

Topics - WinSetLangInfo

Select an item:

Syntax - WinSetLangInfo

```
/*****************************************/
/* This function sets the window Bidi      */
/* attributes and status.                  */
```

```
/*****************************************/

#define INCL_PMBIDI
#include <os2.h>

HWND      hwnd;         /* Window handle */
ULONG     ulEffect;     /* Specifies type of Bidi information to be set */
ULONG     ulData;       /* Data to be set in specified attribute */
ULONG     ulMask;       /* Mask value indicating attribute/status fields to be updated */
ULONG     ulFlags;      /* Flag to determine optional behavior */
ULONG     ulReserved;   /* Reserved */
ULONG     fsuccess;     /* Return value */

   fsuccess  =  WinSetLangInfo ( hwnd, ulEffect, ulData, ulMask,
                          ulFlags, ulReserved );
```

Parameters - WinSetLangInfo

hwnd (HWND) - input
Window handle.

Handle of the window whose attributes or status is to be set. This parameter should be NULLHANDLE when the information being queried is not window-related.

ulEffect (ULONG) - input
Specifies the type of Bidi information to be set.

It determines which type of Bidi-information is set Depending on its value.

LI_BD_CLIP_ATTR
Clipboard bidirectional attributes.

LI_BD_CLIP_CONV_ATTR
The clipboard conversion bidirectional attributes.

LI_BD_PROCESS_ATTR
The process bidirectional attributes.

LI_BD_WND_ATTR
The window Bidi attributes.

LI_BD_WND_STAT
The window Bidi status.

ulData (ULONG) - input
Data to be set in specified attribute.

The value of the specified fields to be set in the Bidi attributes or the Bidi status. These values are contained in the BIDIATTR and the BIDISTAT.

ulMask (ULONG) - input
Mask value indicating attribute or status fields to be updated. This is a bit value that is used to mask the uldata value when the effect is related to a window (LI_BD_WND_ATTR or LI_BD_WND_STAT). Only the bits that are set in ulMask, are enabled for update by the values in ulData.

Depending on the value of ulEffect, ulMask can have one or more of the values in either the Bidi status (see BIDISTATM) or the Bidi attributes (see BIDIATTRM).

ulFlags (ULONG)
Flag to determine optional behavior.

These are flags that determine optional behavior of this function. The following are possible values:

LIF_NO_SENDMSG
No message is sent to the window to inform it of change. This value is valid only when the operation is related to

windows.

LIF_CHILD_INHERIT
: The operation is performed for the window whose handle is specified and all his child windows. This value is valid only when the operation is related to windows.

LIF_WND_REFRESH
: Refresh all the windows whose bidirectional information is being set. This value is valid only when the operation is related to windows.

ulReserved (ULONG)
: Is reserved and must be Zero.

Return Values - WinSetLangInfo

fSuccess (ULONG) - return
: Success indicator.

TRUE
: Setting of the Bidi attribute succeeded.

FALSE
: An error occurred. The WinGetLAstError function may return the following errors:

- PMERR_INVALID_HWND
- PMERR_INVALID_VALUE

Errors - WinSetLangInfo

Notes - WinSetLangInfo

The interpertation of the parameters is dependent on the value of ulEffect. This function is considered "window related" when ulEffect is LI_BD_WND_ATTR or LI_BD_WND_STAT.

When ulEffect is not window related then the value of hwnd must be NULL and the ulMask value is ignored (ie no mask is used, the value provided in ulData is used to set the information).

When ulEffect is window related, the function optionally sends a WM_SETBIDIATTR or WM_SETBIDISTAT message to the window procedure whose information is being set. The window procedure can set the bidi information by itself, and/or perform any additional processing that is required when its bidirectional information is set. When LIF_NO_SENDMSG flag is specified no window message is sent by the function. This can be used by the window procedure itself when it updates its own window-related bidirectional information (so that it does not receive a recursive message).

It is recommended that the LIF_NO_SENDMSG flag is not turned on when the function is called from outside the window procedure whose attributes are being set. Applications should generaly use the default (send the message to the window procedure), so that the bidi aware window procedures can take the appropriate actions when the request to change their bidi attributes is made.

There may be cases where the LIF_WND_REFRESH and/or the LIF_CHILD_INHERIT flags are ignored. This can happen when a window procedure processes any of the messages sent by this function (i.e when the LIF_NO_SENDMSG flag is not specified). In these cases, it is up to the window procedure code to determine whether a redraw operation is requiered and whether child windows should process the message or not.

It is recommended that the LIF_NO_SENDMSG flag is NOT turned on, when the code executes outside the window procedure whose attributes are set.

Applications should generally use the default (send the messege to the window procedure), so that Bidi-aware window procedures can take the appropriate actions when the request to change their Bidi attributes/status is made.

Example - WinSetLangInfo

This example sets the window attributes and those of its children to make their WND and TEXT orientations RTL.

```
#define INCL_PMBIDI
#include <OS2.H>
#include <PMBIDI.H>


VOID SetWindowBidiAttributes(HWND hwnd)
 {
    if (WinSetLangInfo(hwnd,
          LI_BD_WND_ATTR,           /* Type of information to be set is window attributes       */
          BDA_WND_ORIENT_RTL |      /* Data to be set is RTL Wnd and Text orientations          */
          BDA_TEXT_ORIENT_RTL,
          BDAM_WND_ORIENTATION |    /* Masks indicating which attributes to modify              */
          BDAM_TEXT_ORIENTATION,
          LIF_CHILD_INHERIT,       /* Flag to indicate that the operation is to be            */
                                    /* performed for that window and its children              */
          0L))                      /* Reserved to 0                                           */

         WinMessageBox (HWND_DESKTOP,
                        hwnd,
                        "Operation succeeded",
                        "Result",
                         0,
                         MB_OK);
    else
         WinMessageBox (HWND_DESKTOP,
                        hwnd,
                        "Operation failed",
                        "Error",
                         0,
                         MB_OK);

 }
```

Related Functions - WinSetLangInfo

Related Functions

- WinQueryLangInfo

WinSetLangViewer

Topics - WinSetLangViewer

Select an item:

Syntax - WinSetLangViewer

```
/*************************************/
/* This function is used to register    */
/* a 'Language Viewer' window with the  */
/* system.                              */
/*************************************/

#define INCL_PMBIDI
#include <os2.h>

HAB       hab;          /* Anchor block    */
HWND      hwnd;         /* Window Handle   */
ULONG     Codepage;     /* window codepage */
HWND      hwndPrev;     /* return value    */

      hwndPrev  =  WinSetLangViewer( hab, hwnd, Codepage );
```

Parameters - WinSetLangViewer

hab (HAB) - anchor block
Anchor block.

The anchor block handle of the application that registers the Language Viewer.

hwnd (HWND) - input
Window handle.

The window handle of the new Language Viewer window.

Codepage (ULONG) - input
Window codepage.

The codepage for which the "Language Viewer" window is registered. The following codepage values are supported:

864	Arabic PC.
862	Hebrew PC.

Return Values - WinSetLangViewer

hwnd (HWND) - return
Result indicator.

Returns the previous Language Viewer handle for this codepage.

NULLHANDLE
No Language Viewer was previously registered for this codepage.

Other
Previous Language Viewer handle for this codepage.

Errors - WinSetLangViewer

Notes - WinSetLangViewer

The "Language Viewer" window displays control and status information that is useful for users of a specific language.

The "Language Viewer" window is sent WM_LANG_INFO_CHANGED notification message by the system to inform it of significant events that are relevent to users of a specific language.

Example - WinSetLangViewer

This example registers a new Language Viewer window for the system. The current code page is used.

```
#define INCL_PMBIDI
#include <OS2.H>
#include <PMBIDI.H>

HAB    hab;
HMQ    hmq;
ULONG  cp;


VOID SetLangViewerWindow(HWND hwnd)
 {
    cp = WinQueryCp(hmq);       /* Query current code page               */

    if (WinSetLangViewer(hab,   /* Anchor block handle of the application */
                         hwnd,  /* that registers the Language Viewer     */
                                /* Window handle of the new language      */
                         cp))   /* viewer window                          */
                                /* Current code page                      */

         WinMessageBox (HWND_DESKTOP,
                        hwnd,
                        "Setting Language Viewer window succeeded.",
                        "Replacing previous Lang Viewer window",
                         0,
                         MB_OK);
    else
         WinMessageBox (HWND_DESKTOP,
                        hwnd,
                        "Setting Language Viewer window succeeded."
                        "No previous Lang Viewer window",
```

```
                        0,
                        MB_OK);
}
```

Related Functions - WinSetLangViewer

Related Functions

- WinQueryLangViewer

Bidirectional Support GPI Functions

This section contains the functions that are related to GPI bidirectional text processing.

See the *GPI Programming Guide and Reference* for details on commom programming techniques for managing GPI.

GpiQueryBidiAttr

Topics - GpiQueryBidiAttr

Select an item:

Function Syntax
Parameters
Return Values
Example
Related Functions

Syntax - GpiQueryBidiAttr

```
/****************************************/
/* This function queries the bidirectional */
/* of a GPI presentation space.           */
/****************************************/

#define INCL_PMBIDI
#include <os2.h>
```

```
HPS       hps;          /* Presentation space handle */
ULONG     ulBidiAttr;   /* Bidirectional Attributes word */

   ulBidiAttr =  GpiQueryBidiAttr  ( hps );
```

Parameters - GpiQueryBidiAttr

hps - input
Presentation Space handle.

Handle of the GPI presentation space whose bidirectional attributes are being set.

Return Values - WinSetKbdLayer

ulBidiAttr (ULONG)
Bidirectional Attributes word.

Errors - GpiQueryBidiAttr

Notes - GpiQueryBidiAttr

Example - GpiQueryBidiAttr

This example Queries the Bidirectional Attributes of the presentation space, modifed the BDA_TEXTTYPE parameter and sets the new value to be the new bidirectional attributes word of the presentation space.

```
#define INCL_PMBIDI
#include <OS2.H>
#include <PMBIDI.H>


VOID SetBidiAttr(VOID)
 {
 HPS hps;
 ULONG ulBidiAttr;
 ULONG fSuccess;
```

```
hps = WinGetPS(hwnd);

if (hps)
   ulBidiAttr = GpiQueryBidiAttr (hps);

ulBidiAttr &= ~BDAM_TEXTTYPE;
ulBidiAttr |= BDA_TEXTTYPE_VISUAL;

fSuccess = GpiSetBidiAttr (hps, ulBidiAttr);

WinReleasePS(hps);

}
```

Related Functions - GpiQueryBidiAttr

Related Functions

- GpiSetBidiAttr

GpiSetBidiAttr

Topics - GpiSetBidiAttr

Select an item:

Syntax - GpiSetBidiAttr

```
/*************************************/
/* This function sets the Bidirectional */
/* attributes of a presentation space.  */
/*************************************/

#define INCL_PMBIDI
#include <os2.h>

HPS       hps;          /* Presentation space handle */
ULONG     ulBidiAttr;   /* Bidirectional Attributes word */
BOOL      fsuccess;     /* Return value  */
```

```
fsuccess  =  GpiSetBidiAttr  ( hps, BidiAttr);
```

Parameters - GpiSetBidiAttr

hps - input
Presentation Space handle.

Handle of the GPI presentation space whose bidirectional attributes are being set.

ulBidiAttr (ULONG) - input
Bidirectional Attributes word BIDIATTR

The Bidirectional Attributes being set.

Return Values - WinSetKbdLayer

fSuccess (BOOL) - return
Success indicator.

TRUE
Sucessful.

FALSE
An error occurred.

Errors - GpiSetBidiAttr

Notes - GpiSetBidiAttr

The default bidirectional attributes word of the presentation space is set at the time the PS is created by inheriting the process bidirectional attributes word.

Note that for normal and micro presentation spaces, the process bidirectional attributes are inherited when the device context is created, so setting the process bidirectional attribute must be done before the device context is created in order to affect the presentation space which is associated with it.

This function is used to dynamically change the setting of the bidirectional attributes of a GPI PS, thus affecting the Bidirectional text conversions that are performed by GPI.

Example - GpiSetBidiAttr

This example sets the Bidirectional Attributes of the presentation space to the set of attributes in ulBidiAttr. flags bit to zero.

```
#define INCL_PMBIDI
#include <OS2.H>
#include <PMBIDI.H>

HWND   hwnd;


VOID SetBidiAttr(VOID)
 {
 HPS hps;
 ULONG ulBidiAttr = BDA_TEXT_ORIENT_LTR | BDA_TEXTTYPE_VISUAL ;
 ULONG fSuccess;

 hps = WinGetPS(hwnd);

 if (hps)
    fSuccess = GpiSetBidiAttr (hps, ulBidiAttr);

 WinReleasePS(hps);

 }
```

Related Functions - GpiSetBidiAttr

Related Functions

- GpiQueryBidiAttr

Bidirectional Text Conversions

OS/2 provides a set of functions to manage the conversion of bidirectional text between different formats. The text is assumed to be in a form/representation that is described by a set of input values and is converted to another form described by a set of output values.

In order to perform a text conversion, the caller must first call LayoutCreateObject to create a layout object. The layout object is automatically associated with a set of default values, known as layout values, that depend on the locale used to create it.

Currently, the following locale is supported:

- Locale_Arabic
- Locale_Hebrew

Using the handle to the layout object created, the caller can query the current values using LayoutQueryValues and set them, using LayoutSetValues, to the values required for the text transformation.

The section Layout Values describes the standard set of layout values that are supported. It also describes which layout values may be changed after creation of the layout object.

The actual text transformation is done by the LayoutTransformText function, which converts its input buffer according to the values associated with the layout object. LayoutTransformText can also optionally provide some additional information about the transformation performed, such as mappings of the input buffer to the output buffer.

LayoutEditShape is a more specialized text conversion function that allows an application to do Arabic character shape editing in and around a specific location in a buffer.

When an application has finished its text conversions, it can release the resources used by the layout object using LayoutDestroyObject.

Layout Values

The following layout values are associated with a layout object when the locale used with LayoutObjectCreate is Locale_Arabic or Locale_Hebrew.

Value Name	Value Type	Set/Get
ActiveBidirection	BOOL	G
ActiveShapeEditing	BOOL	G
ShapeContextSize	LAYOUT_EDIT_SIZE	G
Cellsize	ULONG	SG
InputMode	BOOL	SG
CallerAllocMem	BOOL	SG
QueryValueSize	ULONG	SG
InOutTextDescrMask	ULONG	SG
InOnlyTextDescr	ULONG	SG
OutOnlyTextDescr	ULONG	SG
Orientation	LAYOUT_TEXT_DESCRIPTOR	SG
TypeOfText	LAYOUT_TEXT_DESCRIPTOR	SG
Swapping	LAYOUT_TEXT_DESCRIPTOR	SG
Numerals	LAYOUT_TEXT_DESCRIPTOR	SG
TextShaping	LAYOUT_TEXT_DESCRIPTOR	SG
WordBreak	LAYOUT_TEXT_DESCRIPTOR	SG

ActiveBidirection (BOOL)

If ActiveBidirection is set to True then the layout object includes knowledge of directional code elements and proper rendering of text will require reordering of directional code elements. Otherwise the layout object does not require any reordering of directional code elements and all code elements will be classified as left_to_right.

The ActiveBidirection value is guaranteed not to change for the life of the layout object.

For Locale_Arabic and Locale_Hebrew, ActiveBidirection is always True.

Active Shape Editing (BOOL)

If ActiveShapeEditing is set to True, then the layout object includes knowledge of context dependent code elements (i.e. an Automatic Shape Determination algorithm) that requires shaping for presentation. If True, the user of a layout object is required to perform some transformation and/or edit shaping prior to rendering the text.

Otherwise, the layout object does no shaping and all code elements may be presented independent of the surrounding characters.

The ActiveShapeEditing value is guaranteed not to change for the life of the layout object.

For Locale_Arabic, ActiveShapeEditing is always True.

For Locale_Hebrew, ActiveShapeEditing is always False.

ShapeContextSize (LAYOUT_EDIT_SIZE)

The ShapeContextSize specifies the size of the context (surrounding code elements) that needs to be accounted for when performing ActiveShapeEditing. The ShapeContextSize is defined as a structure of type LAYOUT_EDIT_SIZE, which defines the number of surrounding code elements that need to be considered when performing edit shaping, i.e., calling the LayoutEditShape function.

When a substring is inserted into a string, the front and back elements define the number of code elements after the substring and the number of code elements before the substring, respectively, that need to be considered when performing edit shaping.

If both front and back elements are set to zero, then no additional context needs to be considered for edit shaping. When ActiveShapeEditing is not set (False), the front and back are guaranteed to be zero (0).

The ShapeContextSize value is guaranteed not to change for the life of the layout object.

For Locale_Arabic, ShapeContextSize.front = ShapeContextSize.back = 3.

For Locale_Hebrew, ShapeContextSize is not relevant.

CellSize (ULONG)

The size of each character cell (is > 1 when each character is followed by one or more attributes; 'CellSize' is 1 when characters occupy consecutive positions in memory).

The default value of CellSize is 1.

InputMode (BOOL)

This value defines whether the implicit algorithm should produce text in its final form (InputMode=FALSE), or if the text should be considered incomplete, as if it is still being input from the keyboard (InputMode=TRUE).

The default value of InputMode is FALSE.

The following values are related to the memory allocation convention used when calling LayoutQueryValues.

CallerAllocMem (BOOL)

If the descriptor CallerAllocMem is set to TRUE, when calling LayoutQueryValues function, it is the responsibility of the caller to allocate storage to store the actual data and freeing this data.

If the descriptor CallerAllocMem is set to FALSE, then LayoutQueryValues allocates storage to store the actual data and the caller is responsible for freeing this data.

For example, if the value of the Orientation text descriptor is being queried, LayoutQueryValues will allocate the memory for the LAYOUT_TEXT_DESCRIPTOR structure and the caller must free the pointer returned using DosFreeMem.

The default value for CallerAllocMem is FALSE;

QueryValueSize (ULONG)

This name is used to query the size of any of the layout values by ORing it with the name of the layout value. In this case the value field will be a pointer to allocation where the size of the layout value will be stored.

Currently, all structures used in OS/2 are of known size, so applications will not need to use this option.

The remaining layout values are all related to describing the format of the input and output buffers. These values define the behaviour of the LayoutTransformText APIs. Two functionally equivalent methods are provided for setting these attributes/descriptors.

InOutTextDescrMask, InOnlyTextDescr and OutOnlyTextDescr

These are three values that allow the caller to use the same bidi attribute masks and bidi attributes that are used in the WinSetLangInfo APIs.

InOutTextDescrMask (ULONG)

This is a bit value that is used to mask the InOnlyTextDescr and OutOnlyTextDescr values. Only the bits that are set in InOutTextDescrMask are enabled for Set/Query of the InOnlyTextDescr and OutOnlyTextDescr values.

InOnlyTextDescr can have one or more of the following values:

- BDAM_TEXTTYPE
- BDAM_TEXT_ORIENTATION
- BDAM_NUMERALS
- BDAM_SYM_SWAP
- BDAM_TEXT_SHAPE
- BDAM_WORD_BREAK
- BDAM_ALL

The default value of InOnlyTextDescr is BDAM_ALL

InOnlyTextDescr (ULONG)

This is the data that is to be used (masked using the InOutTextDescrMask value) to describe the format of the input text.

Possible values are:

- BDA_TEXTTYPE_VISUAL

- BDA_TEXTTYPE_IMPLICIT

- BDA_TEXT_ORIENT_LTR
- BDA_TEXT_ORIENT_RTL
- ORIENTATION_CONTEXT_LTR
- ORIENTATION_CONTEXT_RTL

- BDA_NUMERALS_NOMINAL
- BDA_NUMERALS_HINDI
- BDA_NUMERALS_CONTEXTUAL
-
- BDA_SYM_SWAP_OFF
- BDA_SYM_SWAP_ON

- BDA_WORDBREAK_OFF
- BDA_WORDBREAK_ON

- BDA_TEXT_DISPLAY_SHAPED
- BDA_TEXT_NOMINAL
- BDA_TEXT_INITIAL
- BDA_TEXT_MIDDLE
- BDA_TEXT_FINAL
- BDA_TEXT_ISOLATED

For Locale_Arabic, the default value of InOnlyTextDescr is :

```
BDA_TEXTTYPE_IMPLICIT | BDA_TEXT_ORIENT_LTR | BDA_NUMERALS_NOMINAL |
BDA_SYM_SWAP_OFF | BDA_WORDBREAK_OFF | BDA_TEXT_NOMINAL
```

For Locale_Hebrew, the default value of InOnlyTextDescr is :

```
BDA_TEXTTYPE_IMPLICIT | BDA_TEXT_ORIENT_CONTEXT | BDA_NUMERALS_NOMINAL |
BDA_SYM_SWAP_OFF | BDA_WORDBREAK_OFF | BDA_TEXT_SAVE_SHAPED
```

OutOnlyTextDescr (ULONG)

This is the data that is to be used (masked using the InOutTextDescrMask value) to describe the format of the output text.

Possible values are the same as for InOnlyTextDescr described above.

For Locale_Arabic and Locale_Hebrew, the default value of OutOnlyTextDescr is :

```
BDA_TEXTTYPE_VISUAL | BDA_TEXT_ORIENT_LTR | BDA_NUMERALS_NOMINAL |
BDA_SYM_SWAP_OFF | BDA_WORDBREAK_OFF | BDA_TEXT_DISPLAY_SHAPED
```

The following code fragment shows how an application would set the orientation and type of text using this method:

```
LAYOUT_VALUES     layout[4];
ULONG             ulMask,ulInAttr,ulOutAttr;
     .
     .
     .
                                                        /* Set mask */
ulData          = BDAM_TEXTTYPE | BDAM_TEXT_ORIENTATION;
layout[0].name  = InOutTextDescrMask;
layout[0].value = &ulData;

                                             /* Set input descriptor */
ulInAttr        = BDA_TEXTTYPE_IMPLICIT | BDA_TEXT_ORIENT_RTL;
layout[1].name  = InOnlyTextDescr;
layout[1].value = &ulInAttr;

                                            /* Set output descriptor */
ulOutAttr       = BDA_TEXTTYPE_VISUAL   | BDA_TEXT_ORIENT_LTR;
layout[2].name  = OutOnlyTextDescr;
```

```
layout[2].value = &ulOutAttr;

                                                /* Last value in array */
layout[3].name  = 0;
                                    /* Set the values in the layout object */
 RC = LayoutSetValues(hlo,layout,&index);
```

Using LayoutTextDescriptors

Using the following values, which are of type LAYOUT_TEXT_DESCRIPTOR, the caller can set the input and output values of one or more descriptors. This has exactly the same effect as using the previous method : the choice of which method to use is a matter of style.

Orientation (LAYOUT_TEXT_DESCRIPTOR)

The descriptor Orientation specifies the global directional text orientation:

ORIENTATION-LTR-left-to-right global orientation

ORIENTATION-RTL-right-to-left global orientation

ORIENTATION-CONTEXTUAL-contextual global orientation

When ORIENTATION-CONTEXTUAL is set, the Orientation setting is determined according to the direction of the first significant code element.

TypeOfText (LAYOUT_TEXT_DESCRIPTOR)

The TypeOfText descriptor specifies the directional ordering of the directional text in the buffer:

TEXT-VISUAL-code elements are stored in visually ordered segments which can be rendered as is. Bidirectional code elements will have to be previously reordered for a specific orientation.

TEXT-IMPLICIT-code elements are stored in logically ordered segments.

Swapping (LAYOUT_TEXT_DESCRIPTOR)

The Swapping descriptor specifies whether symmetric swapping is applied to the text:

SWAPPING- The text conforms to symmetric swapping

NO_SWAPPING- The text does not conform to symmetric swapping

Numerals (LAYOUT_TEXT_DESCRIPTOR)

The Numerals descriptor specifies the shaping of numerals recognized by the layout object:

NUMERALS_NOMINAL - Nominal shaping of numerals using the portable character set (i.e. Arabic numerals).

NUMERALS_NATIONAL - National shaping of numeral based on the script of the locale associated with the layout object (i.e. for Arabic, Hindi numerals are used).

NUMERALS_CONTEXTUAL - Contextual shaping of numeral, depending on the context (script) of surrounding text (e.g. Hindi numbers in Arabic text and Arabic numbers otherwise). Contextual shaping of numerals can only be done if TypeOfText is TEXT_IMPLICIT.

Text shaping (LAYOUT_TEXT_DESCRIPTOR)

TextShaping: specifies the characteristics of the in/out text

TEXT_SHAPED - The text contains presentation form shapes as defined by the locale

TEXT_NOMINAL - The text has no presentation form shapes as defined by the locale.

TEXT_INITIAL - The visual text has only initial shapes.

TEXT_MIDDLE - The visual text has only middle shapes.

TEXT_FINAL - The visual text has only final shapes.

TEXT_ISOLATED - The visual text has only isolated shapes.

WordBreak (LAYOUT_TEXT_DESCRIPTOR)

The WordBreak descriptor specifies whether bidirectional should factor word recognition or not.

BREAK _ WordBreak on, word by word reordering (i.e. the text reordering is dependent on the level (per the bidirectional algorithm) of word separators).

NO_BREAK _ word break off, segment reordering.

This descriptor is significant only if TypeOfText is TEXT_IMPLICIT.

The following code fragment shows how an application would set the orientation and type of text using this method. This has exactly the same effect as the previous example :

```
LAYOUT_VALUES     layout[2];
LAYOUT_TEXT_DESCRIPTOR descr;
            .
            .
            .
                            /* Put the descr record in the layout array */
layout[0].name  = TypeOfText | Orientation;
layout[0].value = &descr;

                                                  /* Last value in array */
layout[1].name  = 0;
                                        /* Initialize the input value */
descr.in  = TEXT_IMPLICIT | ORIENTATION_RTL;
                                        /* Initialize the output value */
descr.out = TEXT_VISUAL   | ORIENTATION_LTR;
                                   /* Set the values in the layout object */
 RC = LayoutSetValues(hlo,layout,&index);
```

Layout Functions

Bidirectional applications use bidirectional text. The bidirectional text is identified by the bidirectional attributes. Since applications use different sets of bidirectional attributes (and thus, different representations of bidirectional text), it is necessary to convert this text among representation forms. The most common case is when an application converts the text from the internal (storage) representation to the external (presentation) form.

OS/2 provides a set of Layout functions to manage the conversion of bidirectional text between different formats. The text is assumed to be in a form/representation that is described by a set of input values and is converted to another form described by a set of output values.

LayoutCreateObject

Topics - LayoutCreateObject

Select an item:

Syntax - LayoutCreateObject

```
/*****************************************/
/* This function creates and initializes a  */
/* layout object associated with the locale */
/* specified by locale_name.                */
/*****************************************/

#define INCL_LAYOUT
#include <layout.h>

PUCHAR           locale_name;  /* Name of the locale used */
PLAYOUT_OBJECT   plh;          /* Pointer to a valid layout object */
APIRET           RC;           /* Return value */

     RC  =  LayoutCreateObject ( locale_name, plh );
```

Parameters - LayoutCreateObject

locale_name (PUCHAR) - input
Name of the locale used.

This argument specifies the type of locale used, the following are possible values:

Locale_Arabic	for Arabic support.
Locale_Hebrew	for Hebrew support.

plh (PLAYOUT_OBJECT) - input
Pointer to a valid layout object.

The value of this argument points to a valid layout object that may be used by other layout functions. The returned layout object is initialized to an initial state that defines the behavior of the layout functions. The default initial state is described in the layout values section.

Return Values - LayoutCreateObject

RC(APIRET) - return
Success indicator.

0
layout object points to a valid handle.

Other
An error occurred.

Errors - LayoutCreateObject

none.

Notes - LayoutCreateObject

The layout object is an opaque object containing all the data and methods necessary to perform the layout operations on context dependent/directional characters of the locale name.

When the LayoutCreateObject function completes without errors, the value of the layout object argument points to a valid layout object that may be used by other layout functions.

The returned layout object is initialized to a default state that defines the behavior of the layout functions. The initial state is locale dependent and is described by the layout values that may be queried using the LayoutQueryValues function.

The layout values of the layout object may be changed using the LayoutSetValues function.

Example - LayoutCreateObject

This example creates a layout object.

```
#include <layout.h>

LAYOUT_OBJECT plh;
ULONG RC;

  RC= LayoutCreateObject (Locale_Arabic,&plh);   /* or: Locale_Hebrew */

  if (RC) { printf("Create Error! !!\n"); exit(0);}
```

Related Functions - LayoutCreateObject

Related Functions

- LayoutDestroyObject
- LayoutEditShape
- LayoutQueryValues
- LayoutSetValues
- LayoutTransformText

LayoutDestroyObject

Topics - LayoutDestroyObject

Select an item:

Syntax - LayoutDestroyObject

```
/*****************************************/
/* This function releases all the resources*/
/* of a layout object obtained by the      */
/* LayoutCreateObject function.            */
/*****************************************/

#define INCL_LAYOUT
#include <layout.h>

LAYOUT_OBJECT   plh;       /* Layout object handle */
APIRET          RC;        /* Return value */

     RC  =  LayoutDestroyObject ( plh );
```

Parameters - LayoutDestroyObject

plh (LAYOUT_OBJECT) - input
Layout object handle.

This argument specifies a layout object returned by the LayoutCreateObject function.

Return Values - LayoutDestroyObject

RC(APIRET) - return
Success indicator.

0
All resources associated with layout object were successfully deallocated.

Other
An error occurred.

Errors - LayoutDestroyObject

Notes - LayoutDestroyObject

none.

Example - LayoutDestroyObject

This example creates and then destroys a layout object.

```
#include <layout.h>
LAYOUT_OBJECT plh;
ULONG RC;

  RC= LayoutCreateObject (Locale_Arabic,&plh);     /* or: Locale_Hebrew */
  if (RC) { printf("Create Error! !!\n"); exit(0);}

  RC = LayoutDestroyObject(plh);
  if (RC) { printf(" DESTROY Error!!!\n"); exit(0);}
```

Related Functions - LayoutDestroyObject

Related Functions

- LayoutCreateObject
- LayoutEditShape
- LayoutQueryValues
- LayoutSetValues
- LayoutTransformText

LayoutEditShape

Topics - LayoutEditShape

Select an item:

Syntax - LayoutEditShape

```
/*****************************************/
/* This function edits the shapes of     */
/* the characters pointed to by the      */
/* index parameters.                     */
/*****************************************/

#define INCL_LAYOUT
#include <layout.h>

LAYOUT_OBJECT  plh;     /* Layout object handle */
BOOL           EditType;/* Type of editing */
PULONG         index;   /* Cursor position - in bytes units */
PUCHAR         InpBuf;  /* Input text   */
PULONG         InpSize; /* Size of InpBuf - number of bytes */
PVOID          OutBuf;  /* Output text  */
PULONG         OutSize; /* Size of OutBuf - number of bytes */
APIRET         RC;      /* Return value */

     RC  =  LayoutEditShape ( plh, EditType, index, InpBuf, InpSize,
                              OutBuf, OutSize);
```

Parameters - LayoutEditShape

plh (LAYOUT_OBJECT) - input
Layout object handle.

This argument specifies a layout object returned by the LayoutCreateObject function.

EditType (BOOL) - input
Type of editing.

The EditType argument specifies the type of edit shaping that is desired. The following are possible values:

EditInput
When EditType specifies EditInput, it will read the current code element as defined by index and any preceding (front) code elements as defined by ShapeContextSize layout value.

EditReplace
When EditType specifies EditReplace, it will read the current code element as defined by index and any surrounding (front and/or back) code as defined by ShapeContexSize layout value.

Note: the direction of the editing (i.e. layout values - Orientation and TypeOfText <VISUAL> is what determines which are the preceding and which are the succeeding code elements.

When ActiveShapeEditing is set (True) the layout object maintains an "EditInput state" that may affect subsequent calls to the LayoutEditShape function with EditInput EditType. The EditInput state is not affected when EditType is set to EditReplace. To reset the EditInput state to its initial state these functions should be called with InpBuf set to NULL. The EditInput state is not affected if any

error occurs within the LayoutEditShape is EditInput.

index (PUCHAR) - input
Cursor position - in bytes units.

On input the index argument specifies an offset to the start of a code element in InpBuf that will be the base for the editing. In addition, the context of surrounding code elements may be considered where the minimum set of code elements needed for the specific context dependent script(s) is identified by the ShapeContextSize layout value. If the set of surrounding code elements as defined by index, InpBuf and InpSize is less then the size of front/back of the ShapeContextSize, the LayoutEditShape functions will assume there is no additional context available. It is the caller's responsibility to provide the minimum context if available. The index argument is in units associated with type of InpBuf, i.e. bytes for LayoutEditShape

InpBuf (PVOID) - input
Input text.

The InpBuf argument specifies the source to be processed. A NULL value with EditInput EditType indicates a request to reset the EditInput state to its initial state.

InpSize (PULONG)
Size of InpBuf - number of bytes.

On input, the InpSize argument specifies the number of code elements to be processed in units associated with the type , i.e. bytes for LayoutEditShape. A value of - 1 indicates that input is delimited by a NULL code element. On return, the value is modified to the actual number of code elements that needed shaping in InpBuf. A value of zero(0) when EditType is EditInput indicates that the EditInput state should be reset to its initial state.

OutBuf (PVOID) - output
Output text.

The OutBuf argument contains the shaped output text. This argument can be specified as a NULL pointer to indicate that no transformed text is required. If NULL, the functions will still return the index and InpSize that specify the amount of text required to be redrawn.

OutSize (PULONG) - output
Size of OutBuf - number of bytes.

On input, the OutSize argument specifies the size of the output buffer in number of bytes. Only the code elements required to be shaped are written into OutBuf. The Output buffer should be large enough to contain the shaped result; otherwise, only partial shaping will be performed. If the ActiveShapeEditing layout value is set(True) the OutBuf should be allocated to contain at least:

number of InpBuf code element

On return, the OutSize argument is modified to the actual number of bytes placed in OutBuf.

When the OutSize argument is specified as zero,the function will calculate the size of an output buffer large enough to contain the transformed text from the input buffer, and the result will be returned in this field. The content of the buffers specifies by InpBuf and OutBuf, and the value of InpSize, will remain unchanged.

Return Values - LayoutEditShape

RC(APIRET) - return
Success indicator.

0
The function completed without errors.

LAYOUT_E2BIG
The output buffer is too small and the source text was not processed. Index and InpSize are not guaranteed on return.

Errors - LayoutEditShape

Notes - LayoutEditShape

The LayoutEditShape provides the shapes of the code element pointed to by index along with any surrounding code elements as defined by the ShapeContextSize layout value of the layout object defined by layout_object. the LayoutEditShape functions are intended to be used when performing editing operations on a single code element, e.g. insertion or deletion of a single code element. The LayoutTransform function should be used when doing string shaping. The LayoutEditShape function does not affect any state maintained by the LayoutTransform functions.

When the LayoutEdiShape funtions completes without errors a zero is returned, index and InpSize return the minimum set of code elements required to be redrawn, and if OutBuf is not NULL the respective shaped code elements are written into OutBuf. Any portion of the the InpBuf may be indicated as needing to be redrawn/shaped. Otherwise, if an error occurs a non-zero value is returned and any output values depend on the error code.

Example - LayoutEditShape

This example shows how to use the LayoutEditShape function to reshape the input characters.

```
#include <layout.h>

LAYOUT_OBJECT plh;

UCHAR InpBuf[30];
UCHAR OutBuf[30];

ULONG InpSize = 30;
ULONG OutSize = 30 ;
LAYOUT_VALUES layout[2];
LAYOUT_TEXT_DESCRIPTOR Descr;
ULONG index ;
ULONG RC;

   RC= LayoutCreateObject (Locale_Arabic,&plh);      /* or: Locale_Hebrew */
   if (RC) { printf("Create Error! !!\n"); exit(0);}

   /* Define the LayoutValues that will need changing */
   layout[0].name= TypeOfText|Orientation;
   layout[0].value=descr;

   /* using the OR operator, we set the bits in the Layout Descriptor
    * to describe the orientation of the buffer and the Type of Text
    * that we want for the output */

   descr.in=TEXT_VISUAL|ORIENTATION_LTR;
   descr.out=TEXT_VISUAL|ORIENTATION_LTR;
   layout[1].name=0;           /* End of change */

   /* Set the LayoutValues   */
     RC=LayoutSetValues(plh,layout,&index);

     if (RC)
     {
           printf("SetValue Error at index %d !!!\n",index);
           exit(0);
      }


  /* Here we point to the character were we want to shape */
  index = 4L;
   RC=LayoutEditShape(plh,
                      TRUE,
                      &index,
                      InpBuf,
```

```
                    &InpSize,
                    OutBuf,
                    &OutSize);

    RC = LayoutDestroyObject(plh);
    if (RC) { printf(" DESTROY Error!!!\n"); exit(0);}
```

Related Functions - LayoutEditShape

Related Functions

- LayoutCreateObject
- LayoutDestroyObject
- LayoutQueryValues
- LayoutSetValues
- LayoutTransformText

LayoutQueryValues

Topics - LayoutQueryValues

Select an item:

Syntax - LayoutQueryValues

```
/*************************************/
/* This function queries the current    */
/* setting of Layout values within a    */
/* layout object.                        */
/*************************************/

#define INCL_LAYOUT
#include <layout.h>

LAYOUT_OBJECT  plh;              /* Layout object handle */
LAYOUT_VALUES  values[3];        /* Array of layout values to be queried */
PULONG         index_returned;   /* Value causing the error */
APIRET         RC;               /* Return value */
```

```
RC  =  LayoutQueryValues ( plh, values, index_returned);
```

Parameters - LayoutQueryValues

plh (LAYOUT_OBJECT) - input
Layout object handle.

This argument specifies a layout object returned by the LayoutCreateObject function.

values (PLAYOUT_VALUES) - input
Array of layout values to be queried.

The name field contains the name of the layout value to be queried, and the value field is a pointer to a location where the layout value is to be stored.

index_returned (PULONG) - output
Value causing the error.

If any values can not be queried, then the value of the one causing the error is returned in the index and a non-zero value is returned.

Return Values - LayoutQueryValues

RC(APIRET) - return
Success indicator.

0
All layout values were successfully queried.

index_returned
The layout value specified by index_returned is unknown or the argument Layout_Object is invalid.

Errors - LayoutQueryValues

Notes - LayoutQueryValues

Each value element of a LAYOUT_VALUES record must contain a pointer to the type of the layout value that is being set or get. That is, if the layout value is for type T, the arguement must be of type T*.

If CallerAllocMem is FALSE (default), LayoutQueryValues will allocate storage for the returned data. It is the caller's responsibility to free the allocated memory with DosFreeMem.

If CallerAllocMem is TRUE, LayoutQueryValues will assume that the layout value points to allocated memory. I.e. the caller is responsible for allocating and freeing the memory for the queried data.

Example - LayoutQueryValues

This example first sets the values of an existing layout object then queries these values using the LayoutQueryValues function.

```
#include <layout.h>

LAYOUT_OBJECT plh;
LAYOUT_VALUES layout[3];
LAYOUT_TEXT_DESCRIPTOR Descr;
PLAYOUT_TEXT_DESCRIPTOR Pdescr ;
BOOL CallerAllocate;
ULONG index;
ULONG RC;



  layout[0].name= CallerAllocMem;

  layout[0].value=&CallerAllocate;

  CallerAllocate = TRUE ;

  layout[1].name = 0 ;

  /* Set the LayoutValues   */
  RC=LayoutSetValues (plh,layout,&index);

  if (RC)
  {
        printf("SetValue Error at index %d !!!\n",index);
        exit(0);
  }

  Pdescr = (PLAYOUT_TEXT_DESCRIPTOR)malloc(sizeof (LAYOUT_TEXT_DESCRIPTOR));
  layout[0].name  =  Orientation;
  layout[0].value =  Pdescr;

  layout[1].name=0;           /* End of change */

  RC = LayoutQueryValues(plh,layout,&index);
  if (RC)
  {
        printf("QueryValue Error at index %d !!!\n",index);
        exit(0);
  }
  printf("Orientation in = %lx  out = %lx",Pdescr->in,Pdescr->out);

  free(Pdescr);

  RC = LayoutDestroyObject(plh);
  if (RC) { printf(" DESTROY Error!!!\n"); exit(0);}

 ************************************************************
```

Related Functions - LayoutQueryValues

Related Functions

- LayoutCreateObject
- LayoutDestroyObject
- LayoutEditShape
- LayoutSetValues
- LayoutTransformText

LayoutSetValues

Topics - LayoutSetValues

Select an item:

Syntax - LayoutSetValues

```
/*****************************************/
/* This function is used to change the     */
/* layout values of a layout Object.       */
/*****************************************/

#define INCL_LAYOUT
#include <layout.h>

LAYOUT_OBJECT  plh,             /* Layout object handle */
PLAYOUT_VALUES values,          /* Array of layout values to be set */
PULONG         index_returned   /* Value causing the error */
APIRET         RC;              /* Return value */

     RC  =  LayoutSetValues ( plh, values, index_returned );
```

Parameters - LayoutSetValues

plh (LAYOUT_OBJECT) - input
Layout object handle.

This argument specifies a layout object returned by the LayoutCreateObject function.

values (PLAYOUT_VALUES) - input
Array of layout values to be set.

The name field contains the name of the layout value to be set, and the value field is a pointer to a location where the layout value is to be stored.

index_returned (PULONG) - output
Value causing the error.

If any values can not be set, then the value of the one causing the error is returned in the index and a non-zero value is returned.

Return Values - LayoutSetValues

RC(APIRET) - return
Success indicator.

0
All layout values were successfully set.

index_returned
The layout value specified by index_returned is unknown or its value is invalid or the argument Layout_Object is invalid.

Errors - LayoutSetValues

Notes - LayoutSetValues

The values are written into the layout object and may affect the behavior of subsequent layout functions. Some layout values do alter internal states maintained by a layout_object.

When the LayoutSetvalues function completes without errors all values will have been set in the Layout_Object and a zero is returned.

Example - LayoutSetValues

This example creates then sets the values for a layout object.

```
#include <layout.h>

LAYOUT_OBJECT plh;
LAYOUT_VALUES layout[2];
LAYOUT_TEXT_DESCRIPTOR Descr;
ULONG index;
ULONG RC;


  RC= LayoutCreateObject (Locale_Arabic,&plh);    /* or: Locale_Hebrew */
  if (RC) { printf("Create Error! !!\n"); exit(0);}
  layout[0].name = TypeOfText ;
  layout[0].value = &Descr ;


  Descr.in  = TEXT_IMPLICIT;
  Descr.out = TEXT_VISUAL;

  layout[1].name = 0 ;
```

```
  /* Set the LayoutValues   */
  RC=LayoutSetValues (plh,layout,&index);


  if (RC)
  {
        printf("SetValue Error at index %d !!!\n",index);
        exit(0);
   }
  RC = LayoutDestroyObject(plh);
  if (RC) { printf(" DESTROY Error!!!\n"); exit(0);}
```

Related Functions - LayoutSetValues

Related Functions

- LayoutCreateObject
- LayoutDestroyObject
- LayoutEditShape
- LayoutQueryValues
- LayoutTransformText

LayoutTransformText

Topics - LayoutTransformText

Select an item:

Syntax - LayoutTransformText

```
/****************************************/
/* This function transforms the Inpbuf     */
/* text according to the current layout    */
/* values.                                 */
/****************************************/

#define INCL_LAYOUT
```

```
#include <layout.h>

LAYOUT_OBJECT  plh;       /* Layout Object handle */
PUCHAR         InpBuf;    /* Input text  */
PULONG         InpSize;   /* Size of InpBuf-number of bytes */
PVOID          OutBuf;    /* Output text */
PULONG         OutSize;   /* Size of outBuf-number of bytes */
PULONG         InpToOut;  /* Source to target index array */
PULONG         OutToInp;  /* Target to source index array */
PUCHAR         BidiLevel; /* Bidirectional levels array */
APIRET         RC;        /* Return value */

     RC  =  LayoutTransformText ( plh, InpBuf, InpSize, OutBuf, OutSize,
                                  InpToOut, OutToInp, BidiLevel);
```

Parameters - LayoutTransformText

plh (LAYOUT_OBJECT) - input
Layout object handle.

The plh argument specifies a layout object returned by the LayoutCreateObject function.

InpBuf (PUCHAR) - input
Input text.

The InpBuf argument specifies the source text to be processed. The InpBuf may not be NULL.

InpSize (PULONG) - input
Size of InpBuf-number of bytes.

The InpSize argument specifies the number of code elements to be processed in units associated with the type, i.e. bytes for LayoutTransform. A value of-1 indicates that input is delimited by a NULL code element. On return, the value is modified to the actual number of code elements processed in InBuf (unless the value in OutSize is zero, in which case, the value of InpSize is not changed).

OutBuf (PVOID) - output
Output text.

The OutBuf argument contains the transformed data. This argument can be specified as a Null pointer to indicate that no transformed data is required.

Outsize (PULONG) - output
Size of outBuf-number of bytes.

On input, the OutSize argument specifies the size of the output buffer in number of bytes. The output buffer should be large enough to contain the transformed result; otherwise, only a partial transformation will be performed. If the ActiveShapeEditing layout value is set (True) the OutBuf is recommended to be allocated to contain at least.

number of code element

On return, the OutSize argument is modified to the actual number of bytes placed in OutBuf.

When the OutSize argument is specified as zero, the function will calculate the size of an output buffer large enough to contain the transformed text, and the result will be returned in this field. The content of the buffers specified by InpBuf and OutBuf, and a value of InpSize, will remain unchanged.

InpToOut (PULONG)
Source to target index array.

If the InpToOut argument is not a Null pointer, it represents an array of values with the same number of code elements as InpBuf. On output, the nth value in InpToOut corresponds to the nth code element in InpBuf.

OutToInp (PULONG) - input
Target to source index array.

If the OutToInp argument is not a NULL pointer, it represents an array of values with the same number of code elements as contained in OutBuf. On output, the nth value in OutToInp corresponds to the nth element in OutBuf. This value is the index in InpBuf which identifies the original code element of the nth element in OutBuf. OutToInp may be specified as NULL if no index array from OutBuf to InpBuf is desired.

BidiLevel (PUCHAR) - input
Bidirectional level array.

If the BidiLevel argument is not a NULL pointer, it represents an array of values with the same number of elements as the source text. The nth value in BidiLevel corresponds to the nth code element in InpBuf. This value is the level of this code element as determined by the bidirectional algorithm. BidiLevel may be specified as NULL if a levels array is not desired.

Return Values - LayoutTransformText

RC(APIRET) - return
Success indicator.

0
The function completed without errors.

LAYOUT_E2BIG
The output buffer is full and the source text was not entirely processed.

Errors - LayoutTransformText

Notes - LayoutTransformText

The LayoutTransform function transforms the InpBuf text according to the current layout values in layout object. Any layout value whose value type is LayoutTextDisriptor describes the attributes of the InpBuf and OutBuf. If the attributes are the same of both InpBuf and OutBuf then a null transformation is done with respect to that specific layout avalue.

The output of this functions may be one or more of the following depending on the setting of the respective arguments:

OutBuf/Outsize
any transform data is stored into OutBuf.

InpToOut
a cross reference from each InpBuf code element to the transformed data.

OutToInp
a cross reference to each InpBuf code element from the transformed data.

BidiLevel
a weighted value that represents the directional level of each InpBuf code element. The level is dependent on the internal directional algorithm of the layout object.

Each of these output arguments maybe NULL to specify that no output is desired for the specific argument, but at least one of them should be set to non-NULL to perform any significant work.

When the size of OutBuf is not large enough to contain the entire transformed text, the input text state at the end of the error condition LAYOUT_E2BIG is returned. To resume the transformation on the remaining text, the application should call the LayoutTransform function with the same layout object, the same InpBuf, and Inpsize is set to zero (0).

Example - LayoutTransformText

This example creates a layout object then defines the values that need to be changed and uses the LayoutTransformText function to change them.

```
#include <layout.h>

LAYOUT_OBJECT plh;

UCHAR InpBuf[30];
UCHAR OutBuf[30];
UCHAR BidiLvl[30];
ULONG ToOutBuf[30];
ULONG ToInpBuf[30];
ULONG InpSize = 30  ;
ULONG OutSize = 30   ;
LAYOUT_VALUES layout[2];
LAYOUT_TEXT_DESCRIPTOR Descr;
ULONG index;
ULONG RC;

    RC= LayoutCreateObject (Locale_Arabic,&plh);  /* or: Locale_Hebrew */
    if (RC) { printf("Create Error! !!\n"); exit(0);}

   /* Define the LayoutValues that will need changing */
    layout[0].name= TypeOfText|Orientation|Numerals|TextShaping;
    layout[0].value = &Descr ;


   /* using the OR operator, we set the bits in the Layout Descriptor,
    * to describe the input string attributes, it is RTL, contains
    * Arabic numerals, and it is in Base shapes */

   Descr.in=TEXT_IMPLICIT|ORIENTATION_RTL|NUMERALS_NOMINAL|TEXT_NOMINAL;


   /* Now let's compose the Text Descriptor for the output string to be
    * LTR, Hindu numeral, and Shaped (Automatic Shape Determination) */

   Descr.out=TEXT_VISUAL|ORIENTATION_LTR| NUMERALS_CONTEXTUAL|TEXT_SHAPED;

    layout[1].name = 0 ;

    /* Set the LayoutValues   */
    RC=LayoutSetValues (plh,layout,&index);

  /* calling the function.. to convert the string */
    RC=LayoutTransformText (plh,
                      InpBuf,
                      &InpSize,
                      OutBuf,
                      &OutSize,
                      ToOutBuf,
                      ToInpBuf,
                      BidiLvl);

    if (RC) { printf("Transform Error %d!!!\n",RC); exit(0);}


  RC = LayoutDestroyObject(plh);
  if (RC) { printf(" DESTROY Error!!!\n"); exit(0);}

 ************************************************************
```

Related Functions - LayoutTransformText

Related Functions

Notices

August 1996

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Copyright Notices

Disclaimers

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department LZKS, 11400 Burnet Road, Austin, TX 78758 U.S.A. Such information may be

available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

CUA
IBM
OS/2
Presentation Manager
Workplace Shell

The following terms are trademarks of other companies:

Adobe (Adobe Systems Incorporated)
